LPIC-1:
Linux Professional Institute Certification
Study Guide

LPIC-1:
Linux Professional Institute Certification
Study Guide

Roderick W. Smith

SYBEX

SYBEX and the SYBEX logo are either registered trademarks or trademarks of SYBEX Inc. in the United States and/or other countries.

Screen reproductions produced with The GIMP, a freely distributed piece of software. (http://www.gimp.org)

The CD interface was created using Macromedia Director, COPYRIGHT 1994, 1997-1999 Macromedia Inc. For more information on Macromedia and Macromedia Director, visit http://www.macromedia.com.

TRADEMARKS: SYBEX has attempted throughout this book to distinguish proprietary trademarks from descriptive terms by following the capitalization style used by the manufacturer.

The author and publisher have made their best efforts to prepare this book, and the content is based upon final release software whenever possible. Portions of the manuscript may be based upon pre-release versions supplied by software manufacturer(s). The author and the publisher make no representation or warranties of any kind with regard to the completeness or accuracy of the contents herein and accept no liability of any kind including but not limited to performance, merchantability, fitness for any particular purpose, or any losses or damages of any kind caused or alleged to be caused directly or indirectly from this book.

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Acknowledgments

Although this book bears my name as author, many other people contributed to its creation. Without their help, this book wouldn't exist, or at best would exist in a lesser form. G. Matthew Rice and Tim Writer contributed material to Chapters 3 and 6. Maureen Adams, the acquisitions editor, helped get the book started. Lori Newman and Dennis Fitzgerald served as the book's production editors, so they oversaw the book as it progressed through all its stages. Dan York was the technical editor, who checked the text for technical errors and omissions—but any mistakes that remain are my own. Judy Flynn, the copy editor, helped keep the text grammatical and understandable. The proofreaders, Nancy Riddiough, Jim Brook, and Candace English, checked the text for typos. I'd also like to thank Neil Salkind and others at Studio B, who helped connect me with Wiley to write this book.

Contents at a Glance

Contents

Table of Exercises

Introduction

Why should you learn about Linux? It's a fast-growing operating system, and it is inexpensive and flexible. Linux is also a major player in the small and mid-sized server field, and it's an increasingly viable platform for workstation and desktop use as well. By understanding Linux, you'll increase your standing in the job market. Even if you already know Windows or Mac OS and your employer uses these systems exclusively, understanding Linux will give you an edge when you are looking for a new job or if you are looking for a promotion. For instance, this knowledge will help you to make an informed decision about if and when you should deploy Linux.

The Linux Professional Institute (LPI) has developed its LPI-1 certification as an introductory certification for people who want to enter careers involving Linux. The exam is meant to certify that an individual has the skills necessary to install, operate, and troubleshoot a Linux system and is familiar with Linux-specific concepts and basic hardware.

The purpose of this book is to help you pass both of the LPIC-1 exams (101 and 102). Because these exams cover basic Linux command-line tools, software management, hardware configuration, filesystems, the X Window System, the boot process, scripts, security, documentation, administration, and networking, those are the topics that are emphasized in this book. You'll learn enough to manage a Linux system and how to configure it for many common tasks. Even after you've taken and passed the LPIC 101 and 102 exams, this book should remain a useful reference.

This book has undergone its own testing and certification by ProCert (`http://procert.com/labs_quicklinks/ql_latm.html`). This means that you can rest assured that the book covers the LPIC objectives.

What Is Linux?

Linux is a clone of the Unix operating system (OS) that has been popular in academia and many business environments for years. Formerly used exclusively on large mainframes, Unix and Linux can now run on small computers—which are actually far more powerful than the mainframes of just a few years ago. Because of its mainframe heritage, Unix (and hence also Linux) scales well to perform today's demanding scientific, engineering, and network server tasks.

Linux consists of a kernel, which is the core control software, and many libraries and utilities that rely on the kernel to provide features with which users interact. The OS is available in many different distributions, which are collections of a specific kernel with specific support programs.

Why Become LPI Certified?

Several good reasons to get your LPI certification exist. The LPI website suggests four major benefits:

Relevance LPI's exam was designed with the needs of Linux professionals in mind. This was done by performing surveys of Linux administrators to learn what they actually need to know to do their jobs.

Quality   The LPI exams have been extensively tested and validated using psychometric standards. The result is an ability to discriminate between competent administrators and those who must still learn more material.

Neutrality   LPI is a nonprofit organization that does not itself market any Linux distribution. This fact removes the motivation to create an exam that's designed as a way to market a particular distribution.

Support   The LPI exams are supported by major players in the Linux world. LPI serves the Linux community.

How to Become LPI Certified

The LPI certification is available to anyone who passes the test. You don't have to work for a particular company. It's not a secret society.

To take an LPI exam, you must first register with LPI to obtain an ID number. You can do this online at `https://www.lpi.org/en/register.html`. Your ID number will be e-mailed to you. With the ID number in hand, you can register for the exam with either of the two firms that administer them: Thomson Prometric and Pearson VUE. The exams can be taken at any Thomson Prometric or Pearson VUE testing center. If you pass, you will get a certificate in the mail saying that you have passed. To find the Thomson Prometric testing center nearest you, call (800) 294-3926. Contact (877) 619-2096 for Pearson VUE information. Alternatively, register online at `http://www.2test.com` for Thomson Prometric or `http://www.vue.com` for Pearson VUE. However you do it, you'll be asked for your name, mailing address, phone number, employer, when and where you want to take the test (i.e., which testing center), and your credit card number (arrangement for payment must be made at the time of registration).

Who Should Buy This Book

Anybody who wants to pass the LPIC-1 exams may benefit from this book. If you're new to Linux, this book covers the material you will need to learn the OS from the beginning, and it continues to provide the knowledge you need up to a proficiency level sufficient to pass the LPIC-1 101 and 102 exams. You can pick up this book and learn from it even if you've never used Linux before, although you'll find it an easier read if you've at least casually used Linux for a few days. If you're already familiar with Linux, this book can serve as a review and as a refresher course for information with which you might not be completely familiar. In either case, reading this book will help you to pass the LPIC exams.

This book is written with the assumption that you know at least a little bit about Linux (what it is, and possibly a few Linux commands). I also assume that you know some basics about computers in general, such as how to use a keyboard, how to insert a floppy disk into a floppy drive, and so on. Chances are you have used computers in a substantial way in the past—perhaps even Linux, as an ordinary user, or maybe you have used Windows or Mac OS. I do *not* assume that you have extensive knowledge of Linux system administration, but if you've done some system administration, you can still use this book to fill in gaps in your knowledge.

As a practical matter, you'll need a Linux system with which to practice and learn in a hands-on way. Although LPIC topic 102 is titled "Linux Installation & Package Management," neither

the exam nor this book focuses on actually installing Linux on a computer from scratch, although some of the prerequisites (such as disk partitioning) are covered. You may need to refer to your distribution's documentation to learn how to accomplish this task.

How This Book Is Organized

This book consists of 10 chapters plus supplementary information: a glossary, this introduction, and the assessment test after the introduction. The chapters are organized as follows:

- Chapter 1, "Linux Command-Line Tools," covers the basic tools you need to interact with Linux. These include shells, redirection, pipes, text filters, regular expressions, the Vi editor, and basic process management commands.

- Chapter 2, "Managing Software," describes the programs you'll use to manage software. Much of this task is centered around the RPM and Debian package management systems. The chapter also covers compiling software from source code and managing shared libraries.

- Chapter 3, "Configuring Hardware," focuses on Linux's interactions with the hardware on which it runs. Specific hardware and procedures for using it include the BIOS, expansion cards, modems, USB devices, sound cards, SCSI devices, hard disk partitions, filesystem creation, and boot loaders.

- Chapter 4, "Managing Files and Filesystems," covers the tools used to manage files and the filesystems that hold them. This includes commands to monitor filesystem health, make filesystems available, and manage disk quotas. On the file side, it includes extended information on file management commands, ownership and permissions, and Linux's standard directory tree.

- Chapter 5, "The X Window System," describes the Linux GUI subsystem. Topics include X configuration, managing GUI logins, and setting up a working GUI environment.

- Chapter 6, "The Boot Process and Scripts," explains how Linux boots up and the scripts it uses to do so. Specific topics include kernel modules (drivers), creating a custom kernel, configuring a boot loader, runlevels, SysV startup scripts, the shell environment, and basic scripting information.

- Chapter 7, "Documentation and Security," covers those two topics. Specific subjects include system documentation, Linux documentation on the Internet, restricting access to the computer, and user-level security.

- Chapter 8, "Administering the System," describes miscellaneous administrative tasks. These include user and group management, tuning user environments, managing log files, running jobs in the future, system backups, and setting the clock.

- Chapter 9, "Basic Networking," focuses on basic network configuration. Topics include TCP/IP basics, setting up Linux on a TCP/IP network, setting up a PPP link to the Internet, running a super server, and managing Linux's printing system.

- Chapter 10, "Managing Servers," summarizes some of the more common and important Linux servers. These include the sendmail mail server, the Apache web server, the NFS and Samba file servers, the BIND DNS server, and the SSH remote login server.

Chapters 1 through 5 cover the LPIC 101 exam, while chapters 6 through 10 cover the LPIC 102 exam. These make up Part I and Part II of the book, respectively.

Each chapter begins with a list of the LPIC objectives that are covered in that chapter. (The LPIC objectives contain gaps in their numbering, and the book doesn't cover the objectives in order. Thus, you shouldn't be alarmed at some of the odd ordering of the objectives within the book.) At the end of each chapter, you'll find a couple of elements you can use to prepare for the exam:

Exam Essentials   This section summarizes important information that was covered in the chapter. You should be able to perform each of the tasks or convey the information requested.

Review Questions   Each chapter concludes with 20 review questions. You should answer these questions and check your answer against the one provided after the questions. If you can't answer at least 80 percent of these questions correctly, go back and review the chapter, or at least those sections that seem to be giving you difficulty.

> **WARNING**
>
> The review questions, assessment test, and other testing elements included in this book are *not* derived from the LPIC exam questions, so don't memorize the answers to these questions and assume that doing this will enable you to pass the exam. You should learn the underlying topic, as described in the text of the book. This will let you answer the questions provided with this book *and* pass the exam. Learning the underlying topic is also the approach that will serve you best in the workplace—the ultimate goal of a certification like LPIC.

To get the most out of this book, you should read each chapter from start to finish and then check your memory and understanding with the chapter-end elements. Even if you're already familiar with a topic, you should skim the chapter; Linux is complex enough that there are often multiple ways to accomplish a task, so you may learn something even if you're already competent in an area.

Bonus CD-ROM Contents

This book comes with a CD-ROM that contains several additional elements. Items available on the CD-ROM include the following:

Book contents as a PDF file   The entire book is available as a fully searchable PDF that runs on all Windows platforms as well as on Linux.

Electronic "flashcards"   The CD-ROM includes 150 questions in "flashcard" format (a question followed by a single correct answer). You can use these to review your knowledge of the LPIC exam objectives.

Sample tests   All of the questions in this book appear on the CD-ROM—including the 30-question assessment test at the end of this introduction and the 200 questions that make up the 20-question review question sections for each chapter. In addition, there are two 50-question bonus exams.

Conventions Used in This Book

This book uses certain typographic styles in order to help you quickly identify important information and to avoid confusion over the meaning of words such as on-screen prompts. In particular, look for the following styles:

- *Italicized text* indicates key terms that are described at length for the first time in a chapter. (Italics are also used for emphasis.)

- A `monospaced font` indicates the contents of configuration files, messages displayed at a text-mode Linux shell prompt, filenames, text-mode command names, and Internet URLs.

- `Italicized monospaced text` indicates a variable—information that differs from one system or command run to another, such as the name of a client computer or a process ID number.

- **`Bold monospaced text`** is information that you're to type into the computer, usually at a Linux shell prompt. This text can also be italicized to indicate that you should substitute an appropriate value for your system. (When isolated on their own lines, commands are preceded by non-bold monospaced $ or # command prompts.)

In addition to these text conventions, which can apply to individual words or entire paragraphs, a few conventions highlight segments of text:

A note indicates information that's useful or interesting but that's somewhat peripheral to the main text. A note might be relevant to a small number of networks, for instance, or it may refer to an outdated feature.

A tip provides information that can save you time or frustration and that may not be entirely obvious. A tip might describe how to get around a limitation or how to use a feature to perform an unusual task.

Warnings describe potential pitfalls or dangers. If you fail to heed a warning, you may end up spending a lot of time recovering from a bug, or you may even end up restoring your entire system from scratch.

EXERCISE 1.1

Exercises

An exercise is a procedure you should try out on your own computer to help you learn about the material in the chapter. Don't limit yourself to the procedures described in the exercises, though! Try other commands and procedures to really learn about Linux.

Sidebars

A sidebar is like a note but longer. The information in a sidebar is useful, but it doesn't fit into the main flow of the text.

🌐 **Real World Scenario**

Real World Scenario

A real world scenario is a type of sidebar that describes a task or example that's particularly grounded in the real world. This may be a situation I or somebody I know has encountered, or it may be advice on how to work around problems that are common in real, working Linux environments.

The Exam Objectives

Behind every computer industry exam you can be sure to find exam objectives—the broad topics in which exam developers want to ensure your competency. The official LPI objectives for the LPIC 101 and 102 exams are listed here. (They're also printed at the start of the chapters in which they're covered.)

> **NOTE**   Exam objectives are subject to change at any time without prior notice and at LPI's sole discretion. Please visit the LPIC Certification page of LPI's website (http://www.lpi.org/en/lpic.html) for the most current listing of exam objectives.

Exam 101

Topic 101: Hardware & Architecture

    1.101.1 Configure Fundamental BIOS Settings

    1.101.3 Configure Modem and Sound cards

    1.101.4 Setup SCSI Devices

    1.101.5 Setup different PC expansion cards

    1.101.6 Configure Communication Devices

    1.101.7 Configure USB devices

Topic 102: Linux Installation & Package Management

Topic 103: GNU & Unix Commands

Topic 104: Devices, Linux Filesystems, Filesystem Hierarchy Standard

Topic 110: The X Window System

Exam 102

Topic 105: Kernel

Topic 106: Boot, Initialization, Shutdown and Runlevels

Topic 107: Printing

Topic 108: Documentation

Topic 109: Shells, Scripting, Programming and Compiling

Topic 111: Administrative Tasks

Topic 112: Networking Fundamentals

Topic 113: Networking Services

1.113.1 Configure and manage inetd, xinetd, and related services

1.113.2 Operate and perform basic configuration of sendmail

1.113.3 Operate and perform basic configuration of Apache

1.113.4 Properly manage the NFS, smb, and nmb daemons

1.113.5 Setup and configure basic DNS services

1.113.7 Set up secure shell (OpenSSH)

Topic 114: Security

1.114.1 Perform security administration tasks

1.114.2 Setup host security

1.114.3 Setup user level security

> **NOTE**   The preceding objective list includes only the basic objective titles. You should consult the complete LPIC exam list to learn what commands, files, and procedures you should be familiar with before taking the exam.

Assessment Test

1. How can you enter the BIOS CMOS setup utility to enable or disable onboard devices?

   A. Press F10 during the POST.

   B. Press Delete during the POST.

   C. Press Ctrl+S during the POST.

   D. It depends on the motherboard model.

2. How can you tell whether your system is using `inetd` or `xinetd` as a super server? (Select all that apply.)

   A. Type **ps ax | grep inetd** and examine the output for signs of `inetd` or `xinetd`.

   B. Type **superserver** to see a report on which super server is running.

   C. Look for the `/etc/inetd.conf` file or `/etc/xinetd.d` subdirectory, which are signs of `inetd` or `xinetd`, respectively.

   D. Examine the `/etc/inittab` file to see which super server is launched by `init`, which is responsible for this task.

3. How does the `lpc` utility for CUPS differ from its counterpart in BSD LPD and LPRng?

   A. The `lpc` utility is unique to CUPS; it doesn't ship with BSD LPR or LPRng.

   B. CUPS doesn't ship with an `lpc` command, but BSD LPD and LPRng do.

   C. CUPS's `lpc` is much more complex than its counterpart in BSD LPD and LPRng.

   D. CUPS's `lpc` is much simpler than its counterpart in BSD LPD and LPRng.

4. How do HOWTO documents and FAQs differ?

   A. HOWTOs are book-length works on specified subjects; FAQs are terse command summaries.

   B. HOWTOs are tutorial documents; FAQs are terse command summaries.

   C. HOWTOs are tutorial documents; FAQs are answers to specific questions.

   D. HOWTOs are book-length works on specified subjects; FAQs are answers to specific questions.

5. Which of the following are required when configuring a computer to use a static IP address? (Select all that apply.)

   A. The IP address of the DHCP server

   B. The hostname of the NBNS server

   C. The computer's IP address

   D. The network mask

6. You want a Linux workstation to mount a remote NFS share automatically whenever it boots. How can you accomplish this goal on the Linux workstation (the NFS client)?

  A. Create a file in /etc/xinetd.d for the NFS export you want to mount, providing information about the export on named lines in the file.

  B. Edit /etc/exports to reference the NFS export, providing the computer name and options in parentheses following the computer name.

  C. Edit /etc/fstab to reference the NFS export rather than a Linux device filename, providing a filesystem type code of nfs and a standard Linux mount point.

  D. Any of the above.

7. Which of the following lines, when entered in /etc/lilo.conf, begins a definition to boot Linux using the /boot/bzImage-2.6.19 kernel when the /boot partition is /dev/hda2?

  A. image=(hd0,1)/bzImage-2.6.19

  B. kernel=/boot/bzImage-2.6.19

  C. image=/boot/bzImage-2.6.19

  D. kernel=(hd0,1)/boot/bzImage-2.6.19

8. What does the number 703 represent in the following /etc/passwd entry?

  george:x:703:100:George Brown:/home/george:/bin/tcsh

  A. The account's human ID (HID) number

  B. The account's process ID (PID) number

  C. The account's group ID (GID) number

  D. The account's user ID (UID) number

9. What does the grep command accomplish?

  A. It creates a pipeline between two programs.

  B. It searches files' contents for a pattern.

  C. It concatenates two or more files.

  D. It displays the last several lines of a file.

10. Which of the following are journaling filesystems for Linux? (Select all that apply.)

  A. HPFS

  B. JFS

  C. Ext2fs

  D. Ext3fs

11. In what file do you set the default Linux runlevel?

  A. /etc/inittab

  B. /etc/inittab.conf

  C. /etc/runlevel

  D. /etc/inetd.conf

12. Which printing systems rely on `/etc/printcap` for printer configuration? (Select all that apply.)

   A. BSD LPD

   B. CUPS

   C. Ghostscript

   D. LPRng

13. You're experiencing sporadic problems with a Secure Shell (SSH) login server—users can sometimes log in and sometimes they can't. What might you try immediately after a failure to help diagnose this problem?

   A. On the server computer, type `http://localhost:631` into a web browser to access the SSH configuration page and check its error subpage for error messages.

   B. Type `diagnose sshd` to run a diagnostic on the SSH server daemon (`sshd`).

   C. Type `tail /var/log/messages` to look for error messages from the server.

   D. Examine the /dev/ssh device file to look for error messages from the server.

14. What is the function of the `~/.profile` file?

   A. It is the user configuration file for the ProFTP server.

   B. It is one of a user's `bash` startup scripts.

   C. It is the user configuration file for the ProFile file manager.

   D. Its presence tells `tcsh` to ignore file modes.

15. After unpacking a source code tarball and configuring the software to compile on your system (as described in its documentation), what command are you likely to need to type to compile the software?

   A. `setup`

   B. `make`

   C. `compile`

   D. `gcc`

16. A system administrator logs onto a home computer and uses Telnet over a cable modem connection to access a work computer. This person then uses `su` to acquire superuser privileges to make some changes to the remote system's configuration. What risks does this procedure pose?

   A. The connection can be tracked by spammers, placing both the local and remote systems' users on spammers' address lists.

   B. If the cable modem connection dies, the Telnet server will be left running, consuming considerable system resources until a reboot.

   C. A virus on the remote system might propagate itself into the individual's home computer, thus infecting it and spreading the virus further.

   D. The individual's user account and the `root` password might be discovered by miscreants with sniffers installed on any intervening system.

17. What role does BIND fill?

    A. It allows programs to use dynamic library files.

    B. It translates between hostnames and IP addresses.

    C. It ties together POP and SMTP servers.

    D. It binds an IP address to a network interface.

18. Which of the following is true of automated X configuration tools?

    A. They create a binary configuration file that cannot be manipulated with a text editor.

    B. They all run in stripped-down GUI mode in order to test the X configuration.

    C. They can usually create a working configuration, but this isn't guaranteed.

    D. They can all create both XFree86 3.3.6 and XFree86 4.0.*x* configuration files.

19. Which utility should you use to rename the file `pumpkin.txt` to `lantern.txt`?

    A. `dd`

    B. `rm`

    C. `cp`

    D. `mv`

20. You want to run a lengthy scientific simulation program, called `simbigbang`, which doesn't require any user interaction; the program operates solely on disk files. If you don't want to tie up the shell from which you run the program, what should you type to run `simbigbang` in the background?

    A. `start simbigbang`

    B. `simbigbang &`

    C. `bg simbigbang`

    D. `background simbigbang`

21. Which of the following commands will install an RPM package called `theprogram-1.2.3-4.i386.rpm` on a computer? (Select all that apply.)

    A. `rpm -Uvh theprogram-1.2.3-4.i386.rpm`

    B. `rpm -i theprogram-1.2.3-4.i386.rpm`

    C. `rpm -U theprogram`

    D. `rpm -e theprogram-1.2.3-4.i386.rpm`

22. What tool can diagnose and fix many common Linux filesystem problems?

    A. `mkfs`

    B. `fsck`

    C. `chkdsk`

    D. `scandisk`

23. How can an ordinary user select which window manager to run? (Select all that apply.)

    A. By editing user configuration files like `.xinitrc` or `.xsession`

    B. By editing system configuration files like `/etc/X11/xinit/xinitrc` or `/etc/X11/xdm/Xsession`

    C. By selecting the window manager from a list presented by a GUI login program

    D. By using the standard `Xselector` program, logging out, and logging back in

24. Which of the following commands displays help on *topic* when typed in a Linux shell? (Select all that apply.)

    A. `manual` *topic*

    B. `man` *topic*

    C. `?` *topic*

    D. `info` *topic*

25. A computer's hardware clock keeps track of the time while the computer is powered off. In what format may this time be stored on an *x*86 Linux system? (Select all that apply.)

    A. Coordinated Universal Time (UTC)

    B. Internet Time

    C. Local time

    D. 12-hour time

26. You want to know what kernel modules are currently loaded. What command would you type to learn this information?

    A. `insmod`

    B. `depmod`

    C. `modprobe`

    D. `lsmod`

27. You want to enable all members of the `music` group to read the `instruments.txt` file, which currently has 0640 (`-rw-r-----`) permissions, ownership by `root`, and group ownership by `root`. How might you accomplish this goal? (Select all that apply.)

    A. Type `chown music instruments.txt` in the file's directory.

    B. Type `chgrp music instruments.txt` in the file's directory.

    C. Type `chgroup music instruments.txt` in the file's directory.

    D. Type `chown .music instruments.txt` in the file's directory.

28. Under which of the following circumstances will a chmod command not work?

    A. The user issuing the command doesn't own the file but does own and have write permission to the directory in which the file resides.

    B. The root user issues the command on a file that resides in a read/write filesystem, although the file itself has no write permissions active.

    C. The owner of the file issues the command, but the file's permissions don't grant the owner write access to the file.

    D. The owner of the file issues the command, but the file resides in a directory to which the owner has read but not write access.

29. Which of the following, when typed in Vi's command mode, saves a file and quits from the program?

    A. `:rq`

    B. `:wq`

    C. `:re`

    D. `:we`

30. In evaluating Linux's compatibility with a computer, which factor is more important for SCSI-based systems than for ATA-based computers?

    A. The rotational velocity of the disks

    B. The master/slave status of the disks

    C. The availability of Linux drivers for the adapter card

    D. The support for the disk model in the hdparm utility

Answers to Assessment Test

1. D.   The procedure for entering the BIOS CMOS utility varies from one motherboard to another. It's usually a single keystroke, such as F10 or Delete, during a critical part of the POST. Consult your motherboard manual for details. Most systems also display a prompt telling you how to get to the BIOS CMOS utility, but you may need to read fast to spot it! See Chapter 3 for more information.

2. A, C.   Examining a process listing (obtained from `ps`) for signs of the super server is the most reliable way to determine which one is actually running. The presence of the super server's configuration file or files (as in option C) is also a good diagnostic, although some older systems that have been upgraded may have both sets of configuration files. There is no standard `superserver` utility to report on which one is used. Most distributions launch the super server through a SysV startup script; the `/etc/inittab` file isn't directly involved in this process, so examining it would be pointless. See Chapter 9 for more information.

3. D.   The `lpc` utility is used to start, stop, change the priority of, and otherwise control jobs in a print queue. CUPS ships with an `lpc` utility, but it's quite rudimentary compared to the `lpc` utilities of BSD LPD and LPRng. Instead, CUPS relies on its Web-based interface and stand-alone utilities to provide the ability to control print jobs. See Chapter 9 for more information.

4. C.   Option C correctly summarizes the nature of both HOWTOs and FAQs, both of which can be found at the Linux Documentation Project (LDP). Although some HOWTOs are fairly lengthy, few or none can legitimately be called "book-length." The Guides at the LDP could be so described, though. The description of FAQs as "terse command summaries" is incorrect; that description better applies to `man` pages. See Chapter 7 for more information.

5. C, D.   The computer's IP address and network mask (aka subnet mask or netmask) are the most critical components in TCIP/IP network configuration. (Additional information you may need to provide on many networks includes the IP address of 1 to 3 DNS servers, the hostname or IP address of a router, and the computer's hostname.) You shouldn't need the IP address of a Dynamic Host Configuration Protocol (DHCP) server—and if a DHCP server is present, chances are you should be using DHCP rather than static IP address assignment. A NetBIOS Name Service (NBNS) server converts between names and IP addresses on NetBIOS networks. The hostname of such a computer is not likely to be a critical configuration element, although you may need to provide this information to Samba for some operations to function correctly when sharing files. See Chapter 9 for more information.

6. C.   Linux treats NFS exports much like local filesystems, so you can specify them in `/etc/fstab` along with local filesystems, although some details (most important, the filesystem identifier and the filesystem type code) are different. The `/etc/xinetd.d` directory contains configuration files for servers launched from the `xinetd` super server; it has nothing to do with NFS client configuration. Option B describes in rough terms how to configure an NFS *server,* not an NFS *client* as the question specifies. See Chapter 10 for more information.

7. C. The `image=` line in `/etc/lilo.conf` identifies a kernel image to be booted using normal Linux filenames, so `/boot/bzImage-2.6.19` is the correct notation. There is no `kernel=` option in LILO's configuration file. The `(hd0,1)` notation in options A and D is a GRUB hard disk identifier; this notation is not used in LILO. Option D also uses both the GRUB disk identifier notation and the `/boot` Linux filesystem specification. See Chapter 3 for more information.

8. D. The third field of `/etc/passwd` entries holds the UID number for the account. Linux doesn't use any standard identifier called a human ID (HID), although the acronym HID stands for human interface device, a class of USB devices. Accounts don't have PID numbers; those belong to running processes. The account's GID number is stored in the fourth field of `/etc/passwd`—100 in this example. See Chapter 3 for more information.

9. B. The `grep` command scans files to find those that contain a specified string or pattern. In the case of text files, it displays the matching line or lines; for binary files, it reports that the file matches the pattern. The method of creating a pipeline involves separating two commands with a vertical bar (`|`). The `grep` command can be used in a pipeline, but it doesn't create one. The command that concatenates files is `cat`, and the command that displays the last several lines of a file is `tail`. See Chapter 1 for more information.

10. B, D. The Journaled Filesystem (JFS) is a journaling filesystem written by IBM for AIX and OS/2 and later ported to Linux. The Third Extended Filesystem (ext3fs) is a journaling filesystem based on the older non-journaling Second Extended Filesystem (ext2fs). The High-Performance Filesystem (HPFS) is a non-journaling filesystem designed by Microsoft for OS/2. See Chapter 3 for more information.

11. A. You set the boot-time runlevel on a line that begins with the code `id` within `/etc/inittab`. The `/etc/inittab.conf` and `/etc/runlevel` files are fictitious. The `/etc/inetd.conf` file has nothing to do with setting the runlevel and doesn't even exist on some systems. See Chapter 6 for more information.

12. A, D. The BSD LPD and LPRng printing systems both define printers in `/etc/printcap`, using similar formats. The newer Common Unix Printing System (CUPS) doesn't rely on `/etc/printcap` for configuration, although it does generate a simple `/etc/printcap` file for the benefit of programs that refer to it to learn print queue names. Ghostscript isn't a complete printing system, and it doesn't use `/etc/printcap`, although other printing systems may call Ghostscript to help process print jobs. See Chapter 9 for more information.

13. C. Log files, such as `/var/log/messages` and sometimes others in `/var/log`, often contain useful information concerning server errors. The `tail` program displays the last few lines of a file, so using it to examine log files immediately after a problem occurs can be a useful diagnostic procedure. The `http://localhost:631` URL accesses the Common Unix Printing System (CUPS) configuration utility, which has nothing to do with SSH. There is no standard `diagnose` utility to help diagnose server problems, and there is no standard `/dev/ssh` file. See Chapter 8 for more information.

14. B. The `~./profile` file is one of several `bash` startup scripts. It has nothing to do with the ProFTP server or the `tcsh` shell. The ProFile file manager mentioned in option C is fictitious. See Chapter 6 for more information.

15. B.   The `make` command consults a file (typically called `Makefile`) that contains instructions on how to compile individual source code files and build a working application from source code. Most Linux programs don't ship with compilation or installation tools called `setup` or `compile`. The `gcc` program is a compiler. Although it's often invoked by `make`, most programs you download from the Internet don't require you to call it directly. See Chapter 2 for more information.

16. D.   Because of Telnet's insecure nature, it should not be used for remote administration, especially not over the Internet—passwords and other sensitive data might be discovered by miscreants with access to intervening systems. Ideally, Telnet should not be installed at all. In theory, spammers could eavesdrop on such a connection, but the provided scenario didn't mention the transfer of e-mail addresses, and unless they're explicitly sent, they wouldn't be easily discovered from this connection alone. If the cable modem link goes down, the Telnet server would probably quickly drop the connection. Even if it didn't, the overhead of a single Telnet connection is quite low and is unlikely to be a problem. Viruses don't propagate via Telnet connections—at least, not unless they're spread through some additional commands typed by the user. See Chapter 7 for more information.

17. B.   BIND is the Berkeley Internet Name Domain—a name server. Normally, BIND runs only on your network's name servers; other computers on your network point to a DNS server that runs BIND or similar software. See Chapter 10 for more information.

18. C.   Like many automated tools, the X configuration tools occasionally produce nonworking files. These files are plain text, and so they can be edited in a text editor. The tools themselves may or may not run in GUI mode, and they may or may not provide a means for you to test the configuration. Most tools work only with XFree86 3.3.6 or XFree86 4.0.*x*, not both. See Chapter 5 for more information.

19. D.   The `mv` utility can be used to rename files as well as move them from one location to another. The `dd` utility is used to copy files to backups, while `rm` is used to remove (delete) files and `cp` copies files. See Chapter 4 for more information.

20. B.   Appending an ampersand (`&`) to a command causes that command to execute in the background. The program so launched still consumes CPU time, but it won't monopolize the shell you used to launch it. The `start` and `background` commands are fictitious. Although `bg` does place a job into the background, it doesn't launch a program that way; it places a process that's been suspended (by pressing Ctrl+Z) into the background. See Chapter 1 for more information.

21. A, B.   The `-Uvh` parameter issues an upgrade command (which installs the program whether or not an earlier version is installed) and creates a series of hash marks to display the command's progress. The `-i` parameter installs the program if it's not already installed but causes no progress display. Option C uses a package name, not a complete filename, and so it will fail to install the package file. The `-e` option removes a package. See Chapter 2 for more information.

22. B.   Option B, `fsck`, is Linux's filesystem check utility. It's similar in purpose to the DOS and Windows CHKDSK and ScanDisk utilities, but these DOS and Windows utilities don't work on Linux filesystems like ext2fs or ReiserFS. Option A, `mkfs`, creates new filesystems; it doesn't diagnose or fix filesystem problems. See Chapter 4 for more information.

23. A, C. User configuration files and login-time options can both be used to select the window manager, depending on the system's configuration. Ordinary users should not have write access to system configuration files, although system administrators can change defaults by editing them. Although a program that changes the default window manager for a user would be handy, such a program is not a standard part of Linux or XFree86. See Chapter 5 for more information.

24. B, D. The correct answers, man and info, are two common Linux help packages. Although ? is a common help command within certain interactive programs, it isn't a help command in bash or other common Linux shells. There is no common command called manual. See Chapter 1 for more information.

25. A, C. Unix systems traditionally store time in UTC (aka Greenwich Mean Time), and Linux may do so as well. Most other *x*86 PC OSs traditionally store time as the local time, however, so Linux also supports this option. Internet Time is an alternative to the 24-hour clock in which the day is broken into 1,000 "beats." Standard PC BIOSs don't support this time format. Likewise, a 12-hour clock isn't terribly useful to computers because it doesn't differentiate A.M. from P.M. See Chapter 8 for more information.

26. D. Typing **lsmod** produces a list of the modules that are currently loaded. The insmod and modprobe programs both load modules—either a single module or a single module and all those upon which it depends, respectively. The depmod command generates the modules.dep file that contains module dependency information. See Chapter 6 for more information.

27. B, D. The chgrp and chown commands can both change the group ownership of a file. The chgrp command takes a group name and a filename as parameters, as in option B. The chown command normally changes a file's owner, but if you provide a group name preceded by a dot (.), as in option D, it changes the group of a file. The chown command shown in option A will change the primary ownership of the file to the music user, if such a user exists on the system; it won't change the group ownership. There is no standard chgroup command, as in option C. See Chapter 4 for more information.

28. A. Only the file's owner and root may change permissions on a file via chmod. Whether the file is writeable by the owner is irrelevant, as is whether the directory in which the file resides is writeable. See Chapter 4 for more information.

29. B. The colon (:) starts ex mode, from which you can enter commands. In ex mode, r includes a file in an existing one, w writes a file, e loads an entirely new file, and q quits the program. Thus, the desired combination is :wq. See Chapter 1 for more information.

30. C. Linux provides generic ATA drivers that work with most ATA adapter cards, albeit at low speed. No such generic drivers are available for SCSI; instead, Linux requires device-specific drivers for any SCSI host adapter it uses. Disk rotational velocity doesn't directly affect hardware compatibility for either SCSI or ATA disks. ATA disks can be set as the master or slave device, but SCSI disks don't use these settings. The hdparm utility can measure SCSI or ATA disk performance and can tune some ATA performance parameters, but that utility doesn't need explicit support for specific disk models. See Chapter 3 for more information.

The LPI 101 Exam (106 Weights)

Chapter

1

Linux Command-Line Tools

THE FOLLOWING LINUX PROFESSIONAL INSTITUTE OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **1.103.1 Work on the command line (weight: 5)**
- ✓ **1.103.2 Process text streams using filters (weight: 6)**
- ✓ **1.103.4 Use streams, pipes, and redirects (weight: 5)**
- ✓ **1.103.5 Create, monitor, and kill processes (weight: 5)**
- ✓ **1.103.6 Modify process execution priorities (weight: 3)**
- ✓ **1.103.7 Search text files using regular expressions (weight: 3)**
- ✓ **1.103.8 Perform basic file editing operations using vi (weight: 1)**

Linux borrows heavily from Unix, and Unix began as a text-based operating system (OS). Unix and Linux retain much of this heritage, which means that to understand how to use and, especially, administer Linux, you must understand at least the basics of its command-line tools. Thus, this book begins with an introduction to Linux *shells* (the programs that accept and interpret text-mode commands) and many of the basic commands and procedures you can use from a shell.

This chapter begins with basic shell information, including shell options and basic procedures for using them. From there, this chapter covers streams, pipes, and redirects, which you can use to shunt input and output between programs or between files and programs. These techniques are frequently combined with text processing using filters—commands you can use to manipulate text without the help of a conventional text editor. Sometimes you must manipulate text in an abstract way, using codes to represent several different types of text. This chapter therefore covers this topic. Frequently, using a full-fledged text editor is necessary, so this chapter describes Vi, a basic but common Linux text editor. Finally, this chapter concludes with a look at managing running programs.

Command-Line Basics

Before you do anything else with Linux, you should understand how to use a Linux shell. Several shells are available, but most provide similar capabilities. Understanding a few basics will take you a long way in your use of Linux, so I describe some of these techniques and commands. You should also understand shell *environment variables,* which are placeholders for data that may be useful to many programs. Finally on the topic of command-line basics, you should know how to get help on commands you're trying to use.

Linux Shell Options

As with many key software components, Linux provides a range of options for shells. A complete list would be quite long, but the more common choices include the following:

bash   The GNU Bourne Again Shell (`bash`) is based on the earlier Bourne shell for Unix but extends it in several ways. In Linux, `bash` is the most common default shell for user accounts, and it's the one emphasized in this book and on the LPI exam.

bsh   The Bourne shell upon which `bash` is based also goes by the name `bsh`. It's not often used in Linux, although the `bsh` command is usually a symbolic link to `bash`.

tcsh   This shell is based on the earlier C shell (csh). It's a fairly popular shell in some circles, but no major Linux distributions make it the default shell. Although it's similar to bash in many respects, some operational details differ. For instance, you don't assign environment variables in the same way in tcsh as in bash.

csh   The original C shell isn't much used on Linux, but if a user is familiar with csh, tcsh makes a good substitute.

ksh   The Korn Shell (ksh) was designed to take the best features of the Bourne shell and the C shell and extend them further. It's got a small but dedicated following among Linux users.

zsh   The Z shell (zsh) takes shell evolution further than the Korn Shell, incorporating features from earlier shells and adding still more.

In addition to these shells, dozens more obscure ones are available. In Linux, most users run bash because it's the default. Some other OSs use csh or tcsh as the default, so if your users have backgrounds on non-Linux Unix-like OSs, they may be more familiar with these other shells. You can change a user's default shell by editing the account, as described in Chapter 8, "Administering the System."

The file /bin/sh is a symbolic link to the system's default shell—normally /bin/bash for Linux. This practice enables you to point to a shell (say, at the start of a simple shell script, as described in Chapter 6, "The Boot Process and Scripts") and be assured that a shell will be called, even if the system's available shells change. This feature is particularly important when developing shell scripts that might be run on other computers, as described in Chapter 6.

Using a Shell

Linux shell use is fairly straightforward for anybody who's used a text-mode OS before: You type a command, possibly including options to it, and the computer executes the command. For the most part, Linux commands are external—that is, they're separate programs from the shell. A few commands are internal to the shell, though, and knowing the distinction can be important. You should also know some of the tricks that can ease use of the command shell—how to have the computer complete a long command or filename, or retrieve a command you've recently run, or edit a command you've recently used (or haven't yet fully entered).

> **NOTE**   One class of commands—those for handling basic file management—are very important but are not described here in great detail. For more information on these commands, consult Chapter 4, "Managing Files and Filesystems."

Using Internal and External Commands

Internal commands are, as you might expect, built into the shell. Most shells offer a similar set of internal commands, but shell-to-shell differences do exist, so consult your shell's man page (as

described later, in "Getting Help") for details, particularly if you're using an exotic shell. Internal commands you're likely to use enable you to perform some common tasks:

Change the working directory   Whenever you're running a shell, you're working in a specific directory. When you refer to a file without providing a complete path to the file, the shell works on the file in the current working directory. (Similar rules apply to many programs.) The cd command changes the current working directory. For instance, typing **cd /home/ sally** changes to the /home/sally directory. The tilde (~) character is a useful shortcut; it stands for your home directory, so **cd ~** will have the same effect as **cd /home/sally** if your home directory is /home/sally.

Display the working directory   The pwd command displays the current working directory.

Display a line of text   The echo command displays the text you enter; for instance, typing **echo Hello** causes the system to display the string Hello. This may seem pointless, but it's useful in scripts (described in Chapter 6), and it can also be a good way to review the contents of environment variables (described later in this chapter, in "Using Environment Variables").

Execute a program   The exec command runs an external program that you specify, as in **exec myprog** to run myprog. In most cases, this is better accomplished by typing the name of the program you want to run. The exec command has one special feature, though: Rather than create a new process that runs alongside the shell, the new process *replaces* the shell. When the new process terminates, it's as if you terminated the shell.

Time an operation   The time command times how long subsequent commands take to execute. For instance, typing **time pwd** tells you how long the system took to execute the pwd command. The time is displayed after the full command terminates. Three times are displayed: total execution time (aka real time), user CPU time, and system CPU time. The final two values tell you about CPU time consumed, which is likely to be much less than the total execution time.

Set options   In its most basic form, set displays a wide variety of options relating to bash operation. These options are formatted much like environment variables, but they aren't the same things. You can pass various options to set to have it affect a wide range of shell operations.

Remove options   The unset command removes an option from those that are displayed or modified by set and used by bash.

Terminate the shell   The exit and logout commands both terminate the shell. The exit command terminates any shell, but the logout option terminates only login shells—that is, those that are launched automatically when you initiate a text-mode login as opposed to those that run in xterm windows or the like.

> **NOTE**   This list is not complete. Later sections of this chapter and later chapters describe some additional internal commands. Consult your shell's documentation for a complete list of its internal commands.

Some of these internal commands are duplicated by external commands that do the same thing, but these external commands aren't always installed on all systems. Even when these external commands are installed, the internal command takes precedence unless you provide the complete path to the external command on the command line, as in typing **/bin/pwd** rather than **pwd**.

🌐 Real World Scenario

Confusion over Internal and External Commands

When duplicate internal and external commands exist, they sometimes produce subtly different results or accept different options. These differences can sometimes cause problems. For instance, consider the pwd command and symbolic links to directories. (Symbolic links are described in more detail in Chapter 4. For now, know that they're files that point to other files or directories and for most intents and purposes act just like the files or directories to which they point when they're accessed.) Suppose you create a symbolic link to /bin within your home directory and then cd into that directory. You then want to know where you are. The pwd command that's internal to bash will produce a different result from the external pwd command:

```
$ pwd
/home/sally/binlink
$ /bin/pwd
/usr/bin
```

As you can see, bash's internal pwd shows the path via the symbolic link, whereas the external command shows the path to which the link points. Sometimes these differences can cause confusion, such as if you read the man page or other documentation that describes one version but you use the other and a difference is important. You may wonder why the command isn't operating as you expect. If in doubt, look up the documentation for, and type the complete path to, the external command to be sure you use it.

When you type a command that's not recognized by the shell as one of its internal commands, the shell checks its *path* to find a program by that name to execute it. The path is a list of directories in which commands can be found. It's defined by the PATH environment variable, as described shortly, in "Using Environment Variables." A typical user account will have about half a dozen or a dozen directories in its path. You can adjust the path by changing the PATH environment variable in a shell configuration file, as described in "Shell Configuration."

You can run programs that aren't on the path by providing a complete path on the command line. For instance, typing **./myprog** runs the myprog program in the current directory, and **/home/arthur/thisprog** runs the thisprog program in the **/home/arthur** directory.

> **WARNING**
>
> The root account should normally have a shorter path than ordinary user accounts. Typically, you'll omit directories that store GUI and other user-oriented programs from root's path in order to discourage use of the root account for routine operations, thus minimizing the risk of security breaches related to buggy or compromised binaries being run by root. Most importantly, root's path should *never* include the current directory (./). Placing this directory in root's path makes it possible for a local miscreant to trick root into running replacements for common programs, such as ls, by having root change into a directory with such a program. Indeed, omitting the current directory from ordinary user paths is also generally a good idea. If this directory must be part of the ordinary user path, it should appear at the *end* of the path so that the standard programs take precedence over any replacement programs in the current directory.

In the case of both programs on the path and those whose complete paths you type as part of the command, the program file must be marked as executable. This is done via the execute bit that's stored with the file. Standard programs are marked executable when they're installed, but if you need to adjust a program's executable status, you can do so with the chmod command, as described in Chapter 4.

Shell Command Tricks

Many users find typing commands to be tedious and error prone. This is particularly true of slow or sloppy typists. For this reason, Linux shells include various tools that can help speed up operations. The first of these is *command completion:* Type part of a command or (as an option to a command) a filename and then press the Tab key. The shell tries to fill in the rest of the command or the filename. If just one command or filename matches the characters you've typed so far, the shell fills it in and places a space after it. If the characters you've typed don't uniquely identify a command or filename, the shell fills in what it can and then stops. Depending on the shell and its configuration, it may beep. If you press the Tab key again, the system responds by displaying the possible completions. You can then type another character or two and, if you haven't completed the command or filename, press the Tab key again to have the process repeat.

The most fundamental Linux commands have fairly short names—mv, ls, set, and so on. Some other commands are much longer, though, such as traceroute or sane-find-scanner. Filenames can also be quite lengthy—up to 255 characters on many filesystems. Thus, command completion can save a lot of time when typing. It can also help you avoid typos.

> **NOTE**
>
> The most popular Linux shells, including bash and tcsh, support command and filename completion. Some older shells, though, don't support this helpful feature.

Another helpful shell shortcut is the *shell history*. The shell keeps a record of every command you type (stored in ~/.bash_history in the case of bash). If you've typed a long

command recently and want to use it again, or use a minor variant of it, you can pull the command out of the history. The simplest way to do this is to press the up arrow key on your keyboard; this brings up the previous command. Pressing the up arrow key repeatedly moves through multiple commands so you can find the one you want. If you overshoot, press the down arrow key to move down the history. The Ctrl+P and Ctrl+N keystrokes double for the up and down arrow keys, respectively.

Another way to use the command history is to search through it. Press Ctrl+R to begin a backward (reverse) search, which is what you probably want, and begin typing characters that should be unique to the command you want to find. The characters you type need not be the ones that begin the command; they can exist anywhere in the command. You can either keep typing until you find the correct command or, after you've typed a few characters, press Ctrl+R repeatedly until you find the one you want. The Ctrl+S keystroke works similarly but searches forward in the command history, which might be handy if you've used a backward search or the up arrow key to look back and have overshot. In either event, if you can't find the command you want or change your mind and want to terminate the search, press Ctrl+G to do so.

Frequently, after finding a command in the history, you want to edit it. The `bash` shell, like many shells, provides editing features modeled after those of the Emacs editor:

Move within the line   Press Ctrl+A or Ctrl+E to move the cursor to the start or end of the line, respectively. The left and right arrow keys will move within the line a character at a time. Ctrl+B and Ctrl+F will do the same, moving backward and forward within a line. Pressing Ctrl plus the left or right arrow keys will move backward or forward a word at a time, as will pressing Esc and then B or F.

Delete text   Pressing Ctrl+D or the Delete key deletes the character under the cursor, while pressing the Backspace key deletes the character to the left of the cursor. Pressing Ctrl+K deletes all text from the cursor to the end of the line. Pressing Ctrl+X and then Backspace deletes all the text from the cursor to the beginning of the line.

Transpose text   Pressing Ctrl+T transposes the character before the cursor with the character under the cursor. Pressing Esc and then T transposes the two words immediately before (or under) the cursor.

Change case   Pressing Esc and then U converts text from the cursor to the end of the word to uppercase. Pressing Esc and then L converts text from the cursor to the end of the word to lowercase. Pressing Esc and then C converts the letter under the cursor (or the first letter of the next word) to uppercase, leaving the rest of the word unaffected.

Invoke an editor   You can launch a full-fledged editor to edit a command by pressing Ctrl+X followed by Ctrl+E. The `bash` shell attempts to launch the editor defined by the `$FCEDIT` or `$EDITOR` environment variable or Emacs as a last resort.

These editing commands are just the most useful ones supported by `bash`; consult its `man` page to learn about many more obscure editing features. In practice, you're likely to make heavy use of command and filename completion, command history, and perhaps a few editing features.

Editing Commands

In this exercise, you'll experiment with your shell's completion and command editing tools. To do so, follow these steps:

1. Log in as an ordinary user.

2. Create a temporary directory by typing **mkdir test**. (Directory and file manipulation commands are described in more detail in Chapter 4.)

3. Change into the test directory by typing **cd test**.

4. Create a few temporary files by typing **touch one two three**. This command creates three empty files named one, two, and three.

5. Type **ls -l t**, and without pressing the Enter key, press the Tab key. The system may beep at you or display two three. If it doesn't display two three, press the Tab key again and it should do so. This reveals that either two or three is a valid completion to your command, because these are the two files in the test directory whose filenames begin with the letter t.

6. Type **h**, and again without pressing the Enter key, press the Tab key. The system should complete the command (**ls -l three**), at which point you can press the Enter key to execute it. (You'll see information on the file.)

7. Press the up arrow key. You should see the **ls -l three** command appear on the command line.

8. Press Ctrl+A to move the cursor to the beginning of the line.

9. Press the right arrow key once and type **es** (without pressing the Enter key). The command line should now read **less -l three**.

10. Press the right arrow key once and press the Delete key three times. The command should now read **less three**. Press the Enter key to execute the command. (Note that you can do so even though the cursor is not at the end of the line.) This invokes the less pager on the three file. Because this file is empty, you'll see a mostly empty screen.

11. Press the Q key to exit from the less pager.

Shell Configuration

Shells, like many Linux programs, are configured through files that hold configuration options in a plain-text format. The bash configuration files are actually bash shell scripts, which are described more fully in Chapter 6. For now, you should know that the ~/.bashrc and ~/.profile files are

the main user configuration files for bash, while /etc/bashrc and /etc/profile are the main global configuration files.

Even without knowing much about shell scripting, you can make simple changes to these files. Edit them in your favorite text editor and change whatever needs changing. For instance, you can add directories to the $PATH environment variable, which takes a colon-delimited list of directories.

> **WARNING** Be careful when changing your bash configuration, and particularly the global bash configuration files. Save a backup of the original file before making changes, and test them immediately by logging in using another virtual terminal. If you spot a problem, revert to your saved copy until you can learn the cause and create a working file.

Using Environment Variables

Environment variables are like variables in programming languages—they hold data to be referred to by the variable name. Environment variables differ in that they're part of the environment of a program, and this environment can be modified by other programs, such as the shell. Programs can rely upon environment variables to set information that can apply to many different programs. For instance, suppose a computer hosts several different Usenet news readers. These programs all need to know what Usenet news server to use, so if they all agree to use an environment variable, such as $NNTPSERVER, to hold this information, you can set this environment variable once as part of your shell startup script and then forget it. You needn't set this feature individually for all the news readers installed on the system.

Chapter 6 describes environment variables and their manipulation in more detail. For the moment, you should know that they can be set in bash using the export command:

```
$ export NNTPSERVER=news.abigisp.com
```

This line sets the $NNTPSERVER environment variable to news.abigisp.com. (When setting an environment variable, you omit the dollar sign, but subsequent references include a dollar sign to identify the environment variable as such.) Thereafter, programs that need this information can refer to the environment variable. In fact, you can do so from the shell yourself, using the echo command:

```
$ echo $NNTPSERVER
news.abigisp.com
```

You can also view the entire environment by typing **env**. The result is likely to be several dozen lines of environment variables and their values. Chapter 6 describes what many of these variables are in more detail.

Getting Help

Linux provides a text-based help system known as man. This command's name is short for *manual*, and its entries (its man pages) provide succinct summaries of what a command, file, or other feature does. For instance, to learn about man itself, you would type **man man**. The result is a description of the man command.

The man utility uses the less pager to display information. This program displays text a page at a time. Press the spacebar to move forward a page, Esc followed by V to move back a page, the arrow keys to move up or down a line at a time, the slash (/) key to search for text, and so on. (Type **man less** to learn all the details, or consult the upcoming section, "Paging Through Files with less.") When you're done, press Q to exit from less and the man page it's displaying.

Linux man pages are actually categorized into several sections, which are summarized in Table 1.1. Sometimes a single keyword has entries in multiple sections; for instance, passwd has entries under both section 1 and section 5. In most cases, man returns the entry in the lowest-numbered section; however, you can force the issue by preceding the keyword by the section number. For instance, typing **man 5 passwd** returns information on the passwd file format rather than the passwd command.

Some programs have moved away from man pages to info pages. The basic purpose of info pages is the same as that for man pages, but info pages use a hypertext format so that you can move from section to section of the documentation for a program. Type **info info** to learn more about this system.

TABLE 1.1   Manual Sections

Section Number	Description
1	Executable programs and shell commands
2	System calls provided by the kernel
3	Library calls provided by program libraries
4	Device files (usually stored in /dev)
5	File formats
6	Games
7	Miscellaneous (macro packages, conventions, etc.)
8	System administration commands (programs run mostly or exclusively by root)
9	Kernel routines

Both man pages and info pages are usually written in a terse style. They're intended as reference tools, not tutorials; they frequently assume basic familiarity with the command, or at least with Linux generally. For more tutorial information, you must look elsewhere, such as this book or the Web. The Linux Documentation Project (http://tldp.org) is a particularly relevant Web-based resource for learning about various Linux topics.

Using Streams, Redirection, and Pipes

Streams, redirection, and *pipes* are some of the more powerful command-line tools in Linux. Linux treats the input to and output from programs as a stream, which is a data entity that can be manipulated. Ordinarily, input comes from the keyboard and output goes to the screen (which in this context can mean a full-screen text-mode login session, an xterm or similar window, or the screen of a remote computer via a remote login session). You can redirect these input and output streams to come from or go to other sources, though, such as files. Similarly, you can pipe the output of one program into another program. These facilities can be great tools to tie together multiple programs.

> Part of the Unix philosophy to which Linux adheres is, whenever possible, to do complex things by combining multiple simple tools. Redirection and pipes help in this task by enabling simple programs to be combined together in chains, each link feeding off of the output of the preceding link.

Types of Streams

To begin understanding redirection and pipes, you must first understand the different types of input and output streams. Three are most important for this topic:

Standard input   Programs accept keyboard input via *standard input,* or stdin. In most cases, this is the data that comes into the computer from its keyboard (or from the keyboard of a remote terminal, a network login client, or the like).

Standard output   Text-mode programs send most data to their users via *standard output* (aka stdout), which is normally displayed on the screen, either in a full-screen text-mode session or in a GUI window such as an xterm. (Fully GUI programs such as GUI word processors don't use standard output for their regular interactions, although they might use standard output to display messages in the xterm from which they were launched. GUI output isn't handled via an output stream in the sense I'm describing here.)

Standard error   Linux provides a second type of output stream, known as *standard error*, or stderr. This output stream is intended to carry high-priority information such as error messages. Ordinarily, standard error is sent to the same output device as standard output, so you can't easily tell them apart. You can redirect one independently of the other, though, which can be handy. For

instance, you can redirect standard error to a file while leaving standard output going to the screen so that you can interact with the program and then study the error messages later.

Internally, programs treat these streams just like data files—they open them, read from or write to the files, and close them when they're done. Put another way, ordinary files are streams from a program's point of view. These streams just happen to be the ones used to interact with users.

Redirecting Input and Output

To redirect input or output, you use symbols following the command, including any options it takes. For instance, to redirect the output of the echo command, you would type something like this:

```
$ echo $NNTPSERVER > nntpserver.txt
```

The result is that the file `nntpserver.txt` contains the output of the command (in this case, the value of the $NNTPSERVER environment variable). Redirection operators exist to achieve several effects, as summarized in Table 1.2.

Most of these redirectors deal with output, both because there are two types of output (standard output and standard error) and because you must be concerned with what to do in case you specify a file that already exists. The most important input redirector is <, which takes the specified file's contents as standard input.

T A B L E 1 . 2   Common Redirection Operators

Redirection Operator	Effect
>	Creates a new file containing standard output. If the specified file exists, it's overwritten.
>>	Appends standard output to the existing file. If the specified file does not exist, it's created.
2>	Creates a new file containing standard error. If the specified file exists, it's overwritten.
2>>	Appends standard error to the existing file. If the specified file does not exist, it's created.
&>	Creates a new file containing both standard output and standard error. If the specified file exists, it's overwritten.
<	Sends the contents of the specified file to be used as standard input.
<<	Accepts text on the following lines as standard input.
<>	Causes the specified file to be used for both standard input and standard output.

> **TIP** A common trick is to redirect standard output or standard error to /dev/null. This file is a device that's connected to nothing; it's used when you want to get rid of data. For instance, if the whine program is generating too many error messages, you might type **whine 2> /dev/null** to run it and discard its error messages.

One redirection operator that requires elaboration is ≪. This operator implements a *here document,* which takes text from the following lines as standard input. Chances are you won't use this redirector on the command line, though, because the following lines *are* standard input, so there's no need to redirect them. Rather, you might use this command as part of a script in order to pass data to a command. Unlike most redirection operators, the text immediately following the ≪ code isn't a filename; instead, it's a word that's used to mark the end of input. For instance, typing **someprog ≪ EOF** causes someprog to accept input until it sees a line that contains *only* the string EOF (without even a space following it).

> **NOTE** Some programs that take input from the command line expect you to terminate input by pressing Ctrl+D. This keystroke corresponds to an end-of-file marker using the American Standard Code for Information Interchange (ASCII).

A final redirection tool is the tee command. This command splits standard input so that it's displayed on standard output and on as many files as you specify. Typically, tee is used in conjunction with data pipes so that a program's output can be both stored and viewed immediately. For instance, to view and store the output of someprog, you might type this:

```
$ someprog | tee output.txt
```

> **NOTE** The vertical bar (|) is the pipe character. It implements a pipe, as described in the next section.

Ordinarily, tee overwrites any files whose names you specify. If you want to append data to these files, pass the -a option to tee.

Piping Data Between Programs

Programs can frequently operate on other programs' outputs. For instance, you might use a text-filtering command (such as the ones described shortly, in "Processing Text Using Filters") to manipulate text output by another program. You can do this with the help of redirection operators; send the first program's standard output to a file and then redirect the second program's standard input to read from that file. This solution is awkward, though, and it involves the creation of a file that you might easily overlook, leading to unnecessary clutter on your system.

The solution is to use data pipes (aka pipelines). A pipe redirects the first program's standard output to the second program's standard input and is denoted by a vertical bar (|):

$ *first | second*

For instance, suppose that *first* generates some system statistics, such as system uptime, CPU use, number of users logged in, and so on. This output might be rather lengthy, so you want to trim it a bit. You might therefore use *second*, which could be a script or command that echoes from its standard input only the information in which you're interested. (The grep command, described in "Using grep," is often used in this role.)

Pipes can be used in long sequences:

$ *first | second | third | fourth | fifth | sixth* [...]

Generating Command Lines

Sometimes you'll find yourself constructing command after command that are similar to each other but not similar enough to enable you to use their normal options to substitute a single command. For instance, suppose you want to remove every file in a directory tree with a name that ends in a tilde (~). (This filename convention denotes backup files created by various text editors.) With a large directory tree, this task can be a daunting one; the usual file-deletion command (rm, described in more detail in Chapter 4) doesn't provide an option to search for and delete every file in a directory tree that matches such a specific criterion. One command that can do the search part of the job, though, is find, which is also described in more detail in Chapter 4. This command displays all the files that match criteria you provide. If you could combine the output of find to create a series of command lines using rm, the task would be solved. In fact, this is precisely the purpose of the xargs command.

The xargs command builds a command from its standard input. The basic syntax for this command is as follows:

xargs [*options*] [*command* [*initial-arguments*]]

The *command* is the command you want to execute, and *initial-arguments* are arguments you want to pass to the command. The *options* are xargs options; they aren't passed to *command*. When you run xargs, it runs *command* once for every word passed to it on standard input, adding that word to the argument list for *command*. If you want to pass multiple options to the command, you can protect them by enclosing the group in quotation marks.

For instance, consider the task of deleting all those backup files, denoted by tilde characters. You could do this by piping the output of find to xargs, which then calls rm:

$ **find ./ -name** "*~" **| xargs rm**

The first part of this command (**find ./ -name** "*~") finds all the files in the current directory (./) or its subdirectories with a name that ends in a tilde (*~). This list is then piped to xargs, which adds each one to its own rm command.

A tool that's similar to xargs in many ways is the backtick (`` ` ``), which is a character to the left of the *1* key on most keyboards. The backtick is *not* the same as the single quote character (`'`), which is located to the right of the semicolon (`;`) on most keyboards.

In any event, text within backticks is treated as a separate command whose results are substituted on the command line. For instance, to delete those backup files, you might type the following command:

```
$ rm `find ./ -name "*~"`
```

Processing Text Using Filters

In keeping with Linux's philosophy of providing small tools that can be tied together via pipes and redirection to accomplish more complex tasks, many simple commands to manipulate text are available. These commands accomplish many tasks of various types, such as combining files, transforming the data in files, formatting text, displaying text, and summarizing data.

> **NOTE** Many of the following descriptions include input file specifications. In most cases, these can be omitted, in which case the utility reads from standard input instead.

File-Combining Commands

The first group of text-filtering commands are those used to combine two or more files into one file. Three important commands in this category are cat, join, and paste, which join files end to end, based on fields in the file, or by merging on a line-by-line basis, respectively.

Combining Files with *cat*

The cat command's name is short for *concatenate,* and this tool does just that: It links together an arbitrary number of files end to end and sends the result to standard output. By combining cat with output redirection, you can quickly combine two files into one:

```
$ cat first.txt second.txt > combined.txt
```

Although cat is officially a tool for combining files, it's also commonly used to display the contents of a short file. If you type just one filename as an option, cat ends up displaying that file. This is a great way to review short files; however, for long files, you're better off using a full-fledged pager command, such as more or less.

You can add options to have cat perform some minor modifications to the files as it combines them:

Display line ends   If you want to see where lines end, add the -E or --show-ends option. The result is a dollar sign ($) at the end of each line.

Number lines   The -n or --number option adds line numbers to the beginning of every line. The -b or --number-nonblank option is similar, but it numbers only lines that contain text.

Minimize blank lines   The -s or --squeeze-blank option compresses groups of blank lines down to a single blank line.

Display special characters   The -T or --show-tabs option displays tab characters as ^I. The -v or --show-nonprinting option displays most control and other special characters using carat (^) and M- notations.

Joining Files by Field with *join*

The join command combines two files by matching up the contents of specified fields within the files. Fields are typically space-separated entries on a line, although you can specify another character as the field separator with the -t *char* option, where *char* is the character you want to use.

The effect of join may best be understood through a demonstration. Consider Listings 1.1 and 1.2, which contain data on telephone numbers—Listing 1.1 shows the names associated with those numbers, while Listing 1.2 shows whether the numbers are listed or unlisted.

Listing 1.1: Demonstration File Containing Telephone Number Account Names

```
555-2397 Beckett, Barry
555-5116 Carter, Gertrude
555-7929 Jones, Theresa
555-9871 Orwell, Samuel
```

Listing 1.2: Demonstration File Containing Telephone Number Listing Status

```
555-2397 unlisted
555-5116 listed
555-7929 listed
555-9871 unlisted
```

You can display the contents of both files using join:

```
$ join listing1.1.txt listing1.2.txt
555-2397 Beckett, Barry unlisted
555-5116 Carter, Gertrude listed
555-7929 Jones, Theresa listed
555-9871 Orwell, Samuel unlisted
```

By default, join uses the first field as the one to match across files. Because Listings 1.1 and 1.2 both placed the phone number in this field, it's the key field in the output. You can specify another field by using the -1 or -2 options to specify the join field for the first or second file, respectively, as in **join -1 3 -2 2 people.txt numbers.txt** to join using the third field in people.txt and the second field in numbers.txt.

The join command can be used at the core of a set of simple customized database manipulation tools using Linux text-manipulation commands. It's very limited by itself, though; for

instance, it requires its two files to have the same ordering of lines. (You can use the `sort` command to ensure this is so.)

Merging Lines with *paste*

The `paste` command merges files line by line, separating the lines from each file with tabs. For instance, using Listings 1.1 and 1.2 again:

```
$ paste listing1.1.txt listing1.2.txt
555-2397 Beckett, Barry 555-2397 unlisted
555-5116 Carter, Gertrude        555-5116 listed
555-7929 Jones, Theresa 555-7929 listed
555-9871 Orwell, Samuel 555-9871 unlisted
```

You might use `paste` to combine data from files that aren't keyed with fields suitable for use by `join`. Of course, to be meaningful, the files' line numbers must be exactly equivalent. Alternatively, you might use `paste` as a quick way to create a two-column output of textual data; however, the alignment of the second column might not be exact if the first column's line lengths aren't exactly even, as shown in the preceding example.

File-Transforming Commands

Many of Linux's text manipulation commands are aimed at transforming the contents of files. These commands don't actually change files' contents, though; like most of these commands, they send the changed file to standard output. You can then pipe this output to another command or redirect it into a new file.

> **NOTE** An important file-transforming command is sed. This command is very complex and is covered later in this chapter, in "Using sed."

Convert Tabs to Spaces with *expand*

Sometimes text files contain tabs but programs that need to process the files don't cope well with tabs; or perhaps you want to edit a text file in an editor that uses a different amount of horizontal space for the tab than the editor that created the file. In such cases, you might want to convert tabs to spaces. The `expand` command does this.

By default, `expand` assumes a tab stop every eight characters. You can change this spacing with the `-t` *num* or `--tabs=`*num* option, where *num* is the tab spacing value.

Display Files in Octal with *od*

Some files aren't easily displayed in ASCII; most graphics files, audio files, and so on use non-ASCII characters that look like gibberish. Worse, these characters can do strange things to your display if you try to view such a file with `cat` or a similar tool. For instance, your font may

change or your console may begin beeping uncontrollably. Nonetheless, you might sometimes want to display such files, particularly if you want to investigate the structure of a data file. You might also want to look at an ASCII file in a way that eliminates certain ambiguities, such as whether a gap between words is a tab or several spaces. In such cases, od (whose name stands for *octal dump*) can help. It displays a file in an unambiguous format—octal (base 8) numbers by default. For instance, consider Listing 1.2 as parsed by od:

```
$ od listing1.2.txt
0000000 032465 026465 031462 033471 072440 066156 071551 062564
0000020 005144 032465 026465 030465 033061 066040 071551 062564
0000040 005144 032465 026465 034467 034462 066040 071551 062564
0000060 005144 032465 026465 034071 030467 072440 066156 071551
0000100 062564 005144
0000104
```

The first field on each line is an index into the file in octal. For instance, the second line begins at octal 20 (16 in base 10) bytes into the file. The remaining numbers on each line represent the bytes in the file. This type of output can be difficult to interpret unless you're well versed in octal notation and perhaps in the ASCII code itself.

Although od is nominally a tool for generating octal output, it can in fact generate many other output formats, such as hexadecimal (base 16), decimal (base 10), and even ASCII with escaped control characters. Consult the man page for od for details on creating these variants.

Sort Files with *sort*

Sometimes you'll create an output file that you want sorted. To do so, you can use a command that's called, appropriately enough, sort. This command can sort in several variant ways, including:

Ignore case   Ordinarily, sort sorts by ASCII value, which differentiates between uppercase and lowercase letters. The -i or --ignore-case option causes sort to ignore case.

Month sort   The -M or --month-sort option causes the program to sort by three-letter month abbreviations (JAN through DEC).

Numeric sort   You can sort by number by using the -n or --numeric-sort option.

Reverse sort order   The -r or --reverse option sorts in reverse order.

Sort field   By default, sort uses the first field as its sort field. You can specify another field with the -k *field* or --key=*field* option. (The *field* can, in fact, be two numbered fields separated by commas to sort on multiple fields.)

As an example, suppose you wanted to sort Listing 1.1 by first name. You could do so like this:

```
$ sort -k 3 listing1.1.txt
555-2397 Beckett, Barry
555-5116 Carter, Gertrude
```

```
555-9871 Orwell, Samuel
555-7929 Jones, Theresa
```

The `sort` command supports a large number of additional options, many of them quite exotic. Consult `sort`'s `man` page for details.

Break a File into Pieces with *split*

The `split` command can split a file into two or more files. Unlike most of the text-manipulation commands described in this chapter, this command requires you to enter an output filename—or more precisely, an output filename prefix, to which is added an alphabetic code. You must also normally specify how large you want the individual files to be:

Split by bytes   The `-b` *size* or `--bytes=`*size* option breaks the input file into pieces of *size* bytes. This option can have the usually undesirable consequence of splitting the file mid line.

Split by bytes in line-sized chunks   You can break a file into files of no more than a specified size without breaking lines across files by using the `-C=`*size* or `--line-bytes=`*size* option. (Lines will still be broken across files if the line length is greater than *size*.)

Split by number of lines   The `-l` *lines* or `--lines=`*lines* option splits the file up into chunks with no more than the specified number of lines.

As an example, consider breaking Listing 1.1 into two parts by number of lines:

```
$ split -l 2 listing1.1.txt numbers
```

The result is two files, `numbersaa` and `numbersab`, that together hold the original contents of `listing1.1.txt`.

Reverse the Line Order of a File with *tac*

The `tac` command is named after the `cat` command, but with the letters of its name reversed. This naming is a clue to the command's function: It concatenates files, much like `cat`, but it reverses the order of lines in the files. Here's an example using Listing 1.1:

```
$ tac listing1.1.txt
555-9871 Orwell, Samuel
555-7929 Jones, Theresa
555-5116 Carter, Gertrude
555-2397 Beckett, Barry
```

Translate Characters with *tr*

The `tr` command changes individual characters from standard input. Its syntax is as follows:

```
tr [options] SET1 [SET2]
```

You specify the characters you want replaced in a group (*SET1*) and the characters with which you want them to be replaced as a second group (*SET2*). Each character in *SET1* is replaced with the one at the equivalent position in *SET2*. Here's an example using Listing 1.1:

```
$ tr BCJ bc < listing1.1.txt
555-2397 beckett, barry
555-5116 carter, Gertrude
555-7929 cones, Theresa
555-9871 Orwell, Samuel
```

This example translates some, but not all, of the uppercase characters to lowercase. Note that *SET2* in this example was shorter than *SET1*. The result is that `tr` substitutes the last available letter from *SET2* for the missing letters. In this example, the J in `Jones` became a `c`. The `-t` or `--truncate-set1` option causes `tr` to truncate *SET1* to the size of *SET2* instead.

Another `tr` option is `-d`, which causes the program to delete the characters from *SET1*. When using `-d`, you can omit *SET2* entirely.

The `tr` command also accepts a number of shortcuts, such as `[:alnum:]` (all numbers and letters), `[:upper:]` (all uppercase letters), `[:lower:]` (all lowercase letters), and `[:digit:]` (all digits). You can specify a range of characters by separating them with dashes (-), as in A–M for characters between A and M. Consult `tr`'s `man` page for a complete list of these shortcuts.

Convert Spaces to Tabs with *unexpand*

The `unexpand` command is the logical opposite of `expand`; it converts multiple spaces to tabs. This can help compress the size of files that contain many spaces and could be helpful if a file is to be processed by a utility that expects tabs in certain locations.

Like `expand`, `unexpand` accepts the `-t` *num* or `--tabs=num` option, which sets the tab spacing to once every *num* characters. If you omit this option, `unexpand` assumes a tab stop every eight characters.

Delete Duplicate Lines with *uniq*

The `uniq` command removes duplicate lines. It's most likely to be useful if you've sorted a file and don't want duplicate items. For instance, suppose you want to summarize Shakespeare's vocabulary. You might create a file with all of the Bard's works, one word per line. You could then sort this file using `sort` and pass it through `uniq`. Using a shorter example file containing the text `to be or not to be, that is the question` (one word per line), the result looks like this:

```
$ sort shakespeare.txt | uniq
be
is
not
or
question
```

```
that
the
to
```

   Note that the words `to` and `be`, which appeared in the original file twice, appear only once in the `uniq`-processed version.

File-Formatting Commands

The next three commands, `fmt`, `nl`, and `pr`, reformat the text in a file. The first of these is designed to reformat text files—for instance, if a program's README documentation file uses lines that are too long for your display. The `nl` command numbers the lines of a file, which can be helpful in referring to lines in documentation or correspondence. Finally, `pr` is a print-processing tool; it formats a document in pages suitable for printing.

Reformat Paragraphs with *fmt*

Sometimes text files arrive with outrageously long line lengths, irregular line lengths, or other problems. Depending on the problem, you might be able to cope simply by using an appropriate text editor or viewer to read the file. If you want to clean the file up a bit, though, you can do so with `fmt`. If called with no options (other than the input filename, if you're not having it work on standard input), the program attempts to clean up paragraphs, which it assumes are delimited by two or more blank lines or by changes in indentation. The new paragraph formatting defaults to no more than 75 characters wide. You can change this with the `-*width*`, `-w *width*`, or `--width=*width*` options, though, which set the line length to *width* characters.

Number Lines with *nl*

As described earlier, in "Combining Files with `cat`," you can number the lines of a file with that command. The `cat` line numbering options are limited, though, so if you need to do complex line numbering, `nl` is the tool to use. In its simplest case, you can use `nl` alone to accomplish much the same goal as `cat -b` achieves: numbering all the non-blank lines in a file. You can add many options to `nl`, though, to achieve various special effects, such as:

Body numbering style   You can set the numbering style for the bulk of the lines with the `-b *style*` or `--body-numbering=*style*` option, where *style* is a style format code, described shortly.

Header and footer numbering style   If the text is formatted for printing and has headers or footers, you can set the style for these elements with the `-h *style*` or `--header-numbering=*style*` option for the header and `-f *style*` or `--footer-numbering=*style*` option for the footer.

Page separator   Some numbering schemes reset the line numbers for each page. You can tell `nl` how to identify a new page with the `-d=*code*` or `--section-delimiter=*code*` option, where *code* is a code for the character that identifies the new page.

Line number options for new pages Ordinarily, nl begins numbering each new page with line 1. If you pass the -p or --no-renumber option, though, it doesn't reset the line number with a new page.

Number format You can specify the numbering format with the -n *format* or --number-format=*format* option, where *format* is ln (left justified, no leading zeros), rn (right justified, no leading zeros), or rz (right justified with leading zeros).

The body, header, and footer options enable you to specify a numbering style for each of these page elements:

Number only non-blank lines The default behavior is to number lines that aren't empty. This corresponds to a *style* of t.

Number all lines If you want empty lines to be numbered, specify a *style* of a.

Number no lines To omit line numbers (say, for headers or footers), specify a *style* of n.

Number lines matching a regular expression A *style* of p*REGEXP* numbers only those lines that match the regular expression *REGEXP*. (Regular expressions are described later, in "Using Regular Expressions.")

As an example, suppose you've created a script, buggy, but you find that it's not working as you expect. When you run it, you get error messages that refer to line numbers, so you want to create a version of the script with lines that are numbered for easy reference. You can do so by calling nl with the option to number blank lines (-b a):

```
$ nl -b a buggy > numbered-buggy.txt
```

> **NOTE** Because the input file doesn't have any explicit page delimiters, the output will be numbered in a single sequence; nl doesn't try to impose its own page-length limits.

The numbered-buggy.txt file created by this command isn't useful as a script because of the line numbers that begin each line. You can, however, load it into a text editor or display it with a pager such as less to view the text and see the line numbers along with the commands they contain.

Prepare a File for Printing with *pr*

If you want to print a plain-text file, you may want to prepare it with headers, footers, page breaks, and so on. The pr command was designed to do this. In its most basic form, you pass the command a file:

```
$ pr myfile.txt
```

The result is text formatted for printing on a line printer—that is, pr assumes an 80-character line length in a monospaced font. Of course, you can also use pr in a pipe, either to accept input piped from another program or to pipe its output to another program. (The recipient program might be lpr, which is used to print files, as described in Chapter 9, "Basic Networking.")

By default, `pr` creates output that includes the original text with headers that include the current date and time, the original filename, and the page number. You can tweak the output format in a variety of ways, including:

Multi-column output   Passing the `-numcols` or `--columns=numcols` option creates output with `numcols` columns. Note that `pr` doesn't actually reformat text, though, so if line lengths are too long, they'll be truncated or run over onto multiple lines.

Double-spaced output   The `-d` or `--double-space` option causes double-spaced output from a single-spaced file.

Use form feeds   Ordinarily, `pr` separates pages by using a fixed number of blank lines. This works fine if your printer uses the same number of lines that `pr` expects. If you have problems with this issue, you can pass the `-F`, `-f`, or `--form-feed` option, which causes `pr` to output a form feed character between pages. This works better with some printers.

Set page length   The `-l lines` or `--length=lines` option sets the length of the page in lines.

Set the header text   The `-h text` or `--header=text` option sets the text to be displayed in the header, replacing the filename. To specify a multi-word string, enclose it in quotes, as in `--header="My File"`. The `-t` or `--omit-header` option omits the header entirely.

Set left margin and page width   The `-o chars` or `--indent=chars` option sets the left margin to `chars` characters. This margin size is added to the page width, which defaults to 72 characters and can be explicitly set with the `-w chars` or `--width chars` option.

These options are just the beginning; `pr` supports many more, which are described in its `man` page. As an example of `pr` in action, consider printing a double-spaced and numbered version of a configuration file (say, `/etc/profile`) for your reference. You can do this by piping together `cat` and its `-n` option to generate a numbered output, `pr` and its `-d` option to double-space the result, and `lpr` to print the file:

```
$ cat -n /etc/profile | pr -d | lpr
```

The result should be a printout that might be handy for taking notes on the configuration file. One caveat, though: If the file contains lines that approach or exceed 80 characters in length, the result can be single lines that spill across two lines. The result will be disrupted page boundaries. As a workaround, you could set a somewhat short page length with `-l` and use `-f` to ensure that the printer receives form feeds after each page:

```
$ cat -n /etc/profile | pr -dfl 50 | lpr
```

The `pr` command is built around assumptions about printer capabilities that were reasonable in the early 1980s. It's still useful today, but you might prefer to look into GNU Enscript (`http://people.ssh.fi/mtr/genscript/`). This program has many of the same features as `pr`, but it generates PostScript output that can take better advantage of modern printer features.

File-Viewing Commands

Sometimes you just want to view a file or part of a file. A few commands can help you accomplish this goal without loading the file into a full-fledged editor.

> As described earlier, the cat command is also handy for viewing short files.

Viewing the Starts of Files with *head*

Sometimes all you need to do is see the first few lines of a file. This may be enough to identify what a mystery file is, for instance, or you might want to see the first few entries of a log file to see when that file was started. You can accomplish this goal with the head command, which echoes the first 10 lines of one or more files to standard output. (If you specify multiple filenames, each one's output is preceded by a header to identify it.) You can modify the amount of information displayed by head in two ways:

Specify the number of bytes   The -c *num* or --bytes=*num* option tells head to display *num* bytes from the file rather than the default 10 lines.

Specify the number of lines   You can change the number of lines displayed with the -n *num* or --lines=*num* option.

Viewing the Ends of Files with *tail*

The tail command works just like head, except that tail displays the *last* 10 lines of a file. (You can use the -c/--bytes and -n/--lines options to change the amount of data displayed, just as with head.) This command is useful for examining recent activity in log files or other files to which data may be appended.

The tail command supports several options that aren't present in head and that enable the program to handle additional duties, including:

Track a file   The -f or --follow option tells tail to keep the file open and to display new lines as they're added. This feature is helpful for tracking log files because it enables you to see changes as they're added to the file.

Stop tracking on program termination   The -pid=*pid* option tells tail to terminate tracking (as initiated by -f or --follow) once the process with a process ID (PID) of *pid* terminates. (PIDs are described in more detail later in this chapter, in "Managing Processes.")

Some additional options provide more obscure capabilities. Consult tail's man page for details.

Paging Through Files with *less*

The less command's name is a joke; it's a reference to the more command, which was an early file pager. The idea was to create a better version of more, so the developers called it less.

The idea behind less (and more, for that matter) is to enable you to read a file a screen at a time. When you type **less *filename***, the program displays the first few lines of *filename*. You can then page back and forth through the file:

- Pressing the spacebar moves forward through the file a screen at a time.

- Pressing Esc followed by V moves backward through the file a screen at a time.

- The up and down arrow keys move up or down through the file a line at a time.

- You can search the file's contents by pressing the slash (/) key followed by the search term. For instance, typing **/portable** finds the first occurrence of the string portable after the current position. Typing a slash alone moves to the next occurrence of the search term.

- You can search backward in the file by using the question mark (?) key rather than the slash key.

- You can move to a specific line by typing **g** followed by the line number as, in **g50** to go to line 50.

- When you're done, type **q** to exit from the program.

Unlike most of the programs described here, less can't be readily used in a pipe, except as the final command in the pipe. In that role, though, less is quite useful because it enables you to readily examine lengthy output.

> **NOTE** Although less is quite common on Linux systems and is typically configured as the default text pager, some Unix-like systems use more in this role. Many of less's features, such as the ability to page backward in a file, don't work in more.

One additional less feature can be quite handy: Typing **h** displays less's internal help system. This display summarizes the commands you may use, but it's long enough that you must use the usual less paging features to view it all! When you're done with the help screens, type **q**, just as if you were exiting from viewing a help document with less. This action will return you to your original document.

File-Summarizing Commands

The final text-filtering commands I describe are used to summarize text in one way or another. The cut command takes segments of an input file and sends them to standard output, while the wc command displays some basic statistics on the file.

Extracting Text with *cut*

The cut command extracts portions of input lines and displays them on standard output. You can specify what to cut from input lines in several ways:

By byte   The -b *list* or --bytes=*list* option cuts the specified list of bytes from the input file. (The format of a *list* is described shortly.)

By character   The `-c` *list* or `--characters=`*list* option cuts the specified list of characters from the input file. In practice, this method and the by-byte method usually produce identical results. (If the input file uses a multi-byte encoding system, though, the results won't be identical.)

By field   The `-f` *list* or `--fields=`*list* option cuts the specified list of fields from the input file. By default, a field is a tab-delimited section of a line, but you can change the delimiting character with the `-d` *char* or `--delim=`*char* option, where *char* is the character you want to use to delimit fields. Ordinarily, `cut` echoes lines that don't contain delimiters. Including the `-s` or `--only-delimited` option changes this behavior so that the program does not echo lines that don't contain the delimiter character.

Many of these options take a *list*, which is a way to specify multiple bytes, characters, or fields. This specification is done by number; it can be a single number (such as 4), a closed range of numbers (such as 2-4), or an open range of numbers (such as -4 or 4-). In this final case, all bytes, characters, or fields from the beginning of the line to the specified number or from the specified number to the end of the line are included in the list.

The `cut` command is frequently used in scripts to extract data from some other command's output. For instance, suppose you're writing a script and the script needs to know the hardware address of your Ethernet adapter. This information can be obtained from the `ifconfig` command (described in more detail in Chapter 9):

```
$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0C:76:96:A3:73
          inet addr:192.168.1.3  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:76ff:fe96:a373/64 Scope:Link
          UP BROADCAST NOTRAILERS RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7127424 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5273519 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6272843708 (5982.2 Mb)  TX bytes:1082453585 (1032.3 Mb)
          Interrupt:10 Base address:0xde00
```

Unfortunately, most of this information is extraneous for the desired purpose. The hardware address is the 6-byte hexadecimal number following `HWaddr`. To extract that data, you can combine `grep` (described shortly, in "Using `grep`") with `cut` in a pipe:

```
$ ifconfig eth0 | grep HWaddr | cut -d " " -f 11
00:0C:76:96:A3:73
```

Of course, in a script you would probably assign this value to a variable or otherwise process it through additional pipes. Chapter 6 describes scripts in more detail.

Obtaining a Word Count with *wc*

The `wc` command produces a word count (hence the name), as well as line and byte counts, for a file:

```
$ wc file.txt
  308  2343  15534 file.txt
```

This file contains 308 lines (or, more precisely, 308 newline characters), 2,343 words, and 15,534 bytes. You can limit the output to the newline count, the word count, the byte count, or a character count with the `--lines` (`-l`), `--words` (`-w`), `--bytes` (`-c`), and `--chars` (`-m`) options, respectively. You can also learn the maximum line length with the `--max-line-length` (`-L`) option.

> **NOTE** For an ordinary ASCII file, the character and byte counts will be identical. These values might diverge for files that use multi-byte character encodings.

Using Regular Expressions

Many Linux programs employ *regular expressions*, which are tools for expressing patterns in text. Regular expressions are similar in principle to the wildcards that can be used to specify multiple filenames. At their simplest, regular expressions can be plain text without adornment. Certain characters are used to denote patterns, though. Because of their importance, I describe regular expressions here. I also cover two programs that make heavy use of regular expressions: `grep` and `sed`. These programs search for text within files and permit editing of files from the command line, respectively.

Understanding Regular Expressions

Two forms of regular expression are common: basic and extended. Which form you must use depends on the program; some accept one form or the other, but others can use either type, depending on the options passed to the program. (Some programs use their own minor or major variants on either of these classes of regular expression.) The differences between basic and extended regular expressions are complex and subtle, but the fundamental principles of both are similar.

The simplest type of regular expression is an alphabetic string, such as `Linux` or `HWaddr`. These regular expressions match any string of the same size or longer that contains the regular expression. For instance, the `HWaddr` regular expression matches `HWaddr`, `This is the HWaddr`, and `The HWaddr is unknown`. The real strength of regular expressions comes in the use of non-alphabetic characters, which activate advanced matching rules:

Bracket expressions   Characters enclosed in square brackets (`[]`) constitute bracket expressions, which match any one character within the brackets. For instance, the regular expression `b[aeiou]g` matches the words `bag`, `beg`, `big`, `bog`, and `bug`.

Range expressions   A range expression is a variant on a bracket expression. Instead of listing every character that matches, range expressions list the start and end points separated by a dash (`-`), as in `a[2-4]z`. This regular expression matches `a2z`, `a3z`, and `a4z`.

Any single character   The dot (`.`) represents any single character except for a newline. For instance, `a.z` matches `a2z`, `abz`, `aQz`, or any other three-character string that begins with `a` and ends with `z`.

Start and end of line   The carat (^) represents the start of a line and the dollar sign ($) denotes the end of a line.

Repetition operators   A full or partial regular expression may be followed by a special symbol to denote how many times a matching item must exist. Specifically, an asterisk (*) denotes zero or more occurrences, a plus sign (+) matches one or more occurrences, and a question mark (?) specifies zero or one match. The asterisk is often combined with the dot (as in .*) to specify a match with any substring. For instance, A.*Lincoln matches any string that contains A and Lincoln, in that order—Abe Lincoln and Abraham Lincoln are just two possible matches.

Multiple possible string   The vertical bar (|) separates two possible matches; for instance car|truck matches either car or truck.

Parentheses   Ordinary parentheses (()) surround subexpressions. Parentheses are often used to specify how operators are to be applied, such as surrounding a group of words that are concatenated with the vertical bar to ensure that the words are treated as a group, any one of which may match, without involving surrounding parts of the regular expression.

Escaping   If you want to match one of the special characters, such as a dot, you must escape it—that is, precede it with a backslash (\). For instance, to match a computer hostname (say, twain.example.com), you must escape the dots, as in twain\.example\.com.

The preceding description applies to extended regular expressions. Some details are different for basic regular expressions. In particular, the ?, +, |, (, and) symbols lose their special meaning. To perform the tasks handled by these characters, some programs, such as grep, enable you to recover the functions of these characters by escaping them (say, using \| instead of |). Whether you use basic or extended regular expressions depends on which form the program supports. For programs, such as grep, that support both, you can use either, and which you choose is mostly a matter of personal preference.

Regular expression rules can be confusing, particularly when you're first introduced to them. Some examples of their use, in the context of the programs that use them, will help. The next couple of sections provide such examples.

Using *grep*

The grep command is extremely useful. It searches for files that contain a specified string and returns the name of the file and (if it's a text file) a line of context for that string. The basic grep syntax is as follows:

```
grep [options] regexp [files]
```

The *regexp* is a regular expression, as just described. The grep command supports a large number of options. Some of the more common options enable you to modify the way the program searches files:

Count matching lines   Instead of displaying context lines, grep displays the number of lines that match the specified pattern if you use the -c or --count option.

Specify a pattern input file   The `-f` *file* or `--file=`*file* option takes pattern input from the specified file rather than from the command line.

Ignore case   You can perform a case-insensitive search, rather than the default case-sensitive search, by using the `-i` or `--ignore-case` option.

Search recursively   The `-r` or `--recursive` option searches in the specified directory and all subdirectories rather than simply the specified directory.

Use an extended regular expression   The `grep` command interprets *regexp* as a basic regular expression by default. To use an extended regular expression, you can pass the `-E` or `--extended-regexp` option. Alternatively, you can call `egrep` rather than `grep`; this variant command uses extended regular expressions by default.

A simple example of `grep` uses a regular expression with no special components:

```
$ grep -r eth0 /etc/*
```

This example finds all the files in `/etc` that contain the string `eth0` (the identifier for the first Ethernet device). Because the example includes the `-r` option, it searches recursively, so files in subdirectories of `/etc` are examined as well as those in `/etc` itself. For each matching text file, the line that contains the string is printed.

> **NOTE**   Some files in `/etc` can't be read by ordinary users. Thus, if you type this command as a non-root user, you'll see some error messages relating to `grep`'s inability to open files.

Ramping up a bit, suppose you want to locate all the files in `/etc` that contain the strings `eth0` or `eth1`. You could enter the following command, which uses a bracket expression to specify both variant devices:

```
$ grep eth[01] /etc/*
```

A still more complex example searches all files in `/etc` that contain the hostnames `twain.example.com` or `bronto.pangaea.edu` and, later on the same line, the number `127`. This task requires using several of the regular expression features. Expressed using extended regular expression notation, the command looks like this:

```
$ grep -E "(twain\.example\.com|bronto\.pangaea\.edu).*127" /etc/*
```

This command illustrates another feature you may need to use: shell quoting. Because the shell uses certain characters, such as the vertical bar and the asterisk, for its own purposes, you must enclose certain regular expressions in quotes lest the shell attempt to parse the regular expression as shell commands.

You can use `grep` in conjunction with commands that produce a lot of output in order to sift through that output for the material that's important to you. (Several examples throughout this book use this technique.) For example, suppose you want to find the process ID (PID) of a running

xterm. You can use a pipe to send the result of a ps command (described shortly, in "Managing Processes") through grep, thus:

```
# ps ax | grep xterm
```

The result is a list of all running processes called xterm, along with their PIDs. You can even do this in series, using grep to further restrict the output on some other criterion, which can be useful if the initial pass still produces too much output.

Using *sed*

The sed command directly modifies the contents of files, sending the changed file to standard output. Its syntax can take one of two forms:

```
sed [options] -f script-file [input-file]
sed [options] script-text [input-file]
```

In either case, the *input-file* is the name of the file you want to modify. (Modifications are actually temporary unless you save them in some way, as illustrated shortly.) The script (*script-text* or the contents of *script-file*) is the set of commands you want sed to perform. When passing a script directly on the command line, the *script-text* is typically enclosed in single quote marks. Table 1.3 summarizes a few sed commands that can be used in its scripts.

TABLE 1.3   Common sed Commands

Command	Addresses	Meaning
=	0 or 1	Display the current line number.
a*text*	0 or 1	Append *text* to the file.
i*text*	0 or 1	Insert text into the file.
r *filename*	0 or 1	Append text from filename into the file.
c*text*	range	Replace the selected range of lines with the provided *text*.
s/*regexp*/ *replacement*	range	Replace text that matches the regular expression (*regexp*) with *replacement*.
w *filename*	range	Write the current pattern space to the specified file.
q	0 or 1	Immediately quit the script, but print the current pattern space.
Q	0 or 1	Immediately quit the script.

> **NOTE** Table 1.3 is incomplete; sed is quite complex, and this section merely introduces this tool.

The Addresses column of Table 1.3 requires elaboration: sed commands operate on addresses, which are line numbers. Commands may take no addresses, in which case they operate on the entire file; one address, in which case they operate on the specified line; or two addresses (a range), in which case they operate on that range of lines, inclusive.

In operation, sed looks something like this:

```
$ sed 's/2005/2006/' cal-2005.txt > cal-2006.txt
```

This command processes the input file, `cal-2005.txt`, using sed's s command to replace every occurrence of 2005 with 2006. By default, sed sends the modified file to standard output, so this example uses redirection to send the output to `cal-2006.txt`. The idea in this example is to quickly convert a file created for the year 2005 so that it can be used in 2006. If you don't specify an input filename, sed works from standard input, so it can accept the output of another command as its input.

Although it's conceptually simple, sed is a very complex tool; even a modest summary of its capabilities would fill a chapter. You can consult its man page for basic information, but to fully understand sed, you may want to consult a book on the subject, such as Dale Dougherty and Arnold Robbins's *sed & awk, 2nd Edition* (O'Reilly, 1997).

> **NOTE** Certain sed commands, including the substitution command, are also used in Vi.

Editing Files with Vi

Vi was the first full-screen text editor written for Unix. It's designed to be small and simple. Vi is small enough to fit on tiny, floppy-based emergency boot systems. For this reason alone, Vi is worth learning; you may need to use it in an emergency recovery situation. Vi is, however, a bit strange, particularly if you're used to GUI text editors. To use Vi, you should first understand the three modes in which it operates. Once you understand those modes, you can begin learning about the text-editing procedures Vi implements. This section also examines how to save files and exit from Vi.

> **NOTE** Most Linux distributions actually ship with a variant of Vi known as Vim, or "Vi Improved." As the name implies, Vim supports more features than the original Vi does. The information presented here applies to both Vi and Vim. Most distributions that ship with Vim support launching it by typing **vi**, as if it were the original Vi.

Vi Modes

At any given moment, Vi is running in one of three modes:

Command mode   This mode accepts commands, which are usually entered as single letters. For instance, i and a both enter insert mode, although in somewhat different ways, as described shortly, and o opens a line below the current one.

Ex mode   To manipulate files (including saving your current file and running outside programs), you use ex mode. You enter ex mode from command mode by typing a colon (:), typically directly followed by the name of the ex mode command you want to use. After you run the ex mode command, Vi returns automatically to command mode.

Insert mode   You enter text in insert mode. Most keystrokes result in text appearing on the screen. One important exception is the Esc key, which exits from insert mode back to command mode.

> If you're not sure what mode Vi is in, press the Esc key. This will return you to command mode, from which you can reenter insert mode, if necessary.

Unfortunately, terminology surrounding Vi modes is inconsistent at best. Command mode is sometimes referred to as normal mode, and insert mode is sometimes called edit mode or entry mode, for instance. Ex mode is often not described as a mode at all, but as colon commands.

Basic Text-Editing Procedures

As a method of learning Vi, consider the task of editing /etc/lilo.conf to add a new kernel. Listing 1.3 shows the original lilo.conf file used in this example. If you want to follow along, enter it using a text editor with which you're already familiar and save it to a file on your disk.

Listing 1.3: Sample */etc/lilo.conf* File

```
boot=/dev/sda
map=/boot/map
install=/boot/boot.b
prompt
default=linux
timeout=50
image=/boot/vmlinuz
        label=linux
        root=/dev/sda6
        read-only
```

> **WARNING**   Don't try editing your *real* /etc/lilo.conf file as a learning exercise; a mistake could render your system unbootable the next time you type **lilo**. You might put your test lilo.conf file in your home directory for this exercise.

The first step to using Vi is to launch it and have it load the file. In this example, type **vi lilo.conf** while in the directory holding the file. The result should resemble Figure 1.1, which shows Vi running in a Konsole window. The tildes (˜) down the left side of the display indicate the end of the file. The bottom line shows the status of the last command—an implicit file load command because you specified a filename when launching the program.

Adding a new entry to lilo.conf involves duplicating the lines beginning with the image= line and modifying the duplicates. Therefore, the first editing task is to duplicate these four lines. To do this, follow these steps:

1. Move the cursor to the beginning of the image= line by using the down arrow key; you should see the cursor resting on the i.

> **NOTE**   The h, j, k, and l keys can also be used in place of the left, down, up, and right arrow keys. This is a holdover from the days before all keyboards had arrow keys. Some people prefer to use these keys because they can be faster to use, once the mappings are learned, than the arrow keys.

FIGURE 1.1   The last line of a Vi display is a status line that shows messages from the program.

2. You must now "yank" four lines of text. This term is used much as "copy" is used in most text editors—you copy the text to a buffer from which you can later paste it back into the file. To yank text, you use the yy command, preceded by the number of lines you want to yank. Thus, type **4yy** (*do not* press the Enter key, though). Vi responds with the message `4 lines yanked` on its bottom status line. The dd command works much like yy, but it deletes the lines as well as copying them to a buffer. Both yy and dd are special cases of the y and d commands, respectively, which yank or delete text in amounts specified by the next character, as in dw to delete the next word.

3. Move the cursor to the last line of the file by using the arrow keys.

4. Type **p** (again, without pressing the Enter key). Vi pastes the contents of the buffer starting on the line after the cursor. The file should now have two identical image= stanzas. The cursor should be resting at the start of the second one. If you want to paste the text into the document starting on the line *before* the cursor, use an uppercase P command.

Now that you've duplicated the necessary lines, you must modify one copy to point to your new kernel. To do so, follow these steps:

1. Move the cursor to the v in vmlinuz on the second image= line. You're about to begin customizing this second stanza.

2. Up until now, you've operated Vi in command mode. There are several commands that you can use to enter insert mode. At this point, the most appropriate is R, which enters insert mode so that it is configured for text replacement rather than insertion. If you prefer to insert text rather than overwrite it, you could use i or a (the latter advances the cursor one space, which is sometimes useful at the end of a line). For the purposes of these instructions, type **R** to enter insert mode. You should see -- REPLACE -- appear in the status line.

3. Type the name of a new Linux kernel. For the purposes of this example, let's say you've called it bzImage-2.6.13, so that's what you'd type. This entry should replace vmlinuz.

4. Use the arrow keys to move the cursor to the start of linux on the next line. You must replace this label so that your new entry has its own label.

5. Type a new label, such as **mykernel**. This label should replace the existing linux label.

6. Exit from insert mode by pressing the Esc key.

7. Save the file and quit by typing **:wq**. This is actually an ex mode command, as described shortly. (The ZZ command is equivalent to :wq.)

Many additional commands are available that you might want to use in some situations. Here are some of the highlights:

Case changes   Suppose you need to change the case of a word in a file. Instead of entering insert mode and retyping the word, you can use the tilde (~) key in command mode to change the case. Position the cursor on the first character you want to change and press ~ repeatedly until the task is done.

Undo   To undo any change, type **u** in command mode.

Opening text   In command mode, typing **o** opens text—that is, it inserts a new line immediately below the current one and enters insert mode on that line.

Searches   To search forward for text in a file, type **/** in command mode, followed immediately by the text you want to locate. Typing **?** will search backward rather than forward.

Changes   The c command changes text from within command mode. You invoke it much like the d or y commands, as in cw to change the next word or cc to change an entire line.

Go to a line   The G key brings you to a line that you specify. The H key "homes" the cursor— that is, it moves the cursor to the top line of the screen. The L key brings the key to the bottom line of the screen.

Global replacement   To replace all occurrences of one string by another, type **:%s/*original*/ *replacement***, where *original* is the original string and *replacement* is its replacement. Change % to a starting line number, comma, and ending line number to perform this change on just a small range of lines.

There's a great deal more depth to Vi than is presented here; the editor is quite capable, and some Linux users are very attached to it. Entire books have been written about Vi. Consult one of these, or a Vi web page like http://www.vim.org, for more information.

Saving Changes

To save changes to a file, type **:w** from command mode. This enters ex mode and runs the w ex-mode command, which writes the file using whatever filename you specified when you launched Vi. Related commands enable other functions:

Edit new file   The :e command edits a new file. For instance, **:e /etc/inittab** loads /etc/ inittab for editing. Vi won't load a new file unless the existing one has been saved since its last change or unless you follow :e with an exclamation mark (!).

Include existing file   The :r command includes the contents of an old file in an existing one.

Execute an external command   The ex-mode command :! executes the external command that you specify. For instance, typing **:!ls** runs ls, enabling you to see what files are present in the current directory.

Quit   Use the :q command to quit from the program. As with :e, this command won't work unless changes have been saved or you append an exclamation mark to the command (as in :q!).

You can combine ex commands such as these to perform multiple actions in sequence. For instance, typing **:wq** writes changes and then quits from Vi.

Managing Processes

When you type a command name, that program is run and a *process* is created for it. Knowing how to manage these processes is critical to using Linux. Key details in this task include identifying processes, manipulating foreground and background processes, killing processes, and adjusting process priorities.

Examining Process Lists

One of the most important tools in process management is ps. This program displays processes' status (hence the name, ps). It sports many helpful options, and it's useful in monitoring what's happening on a system. This can be particularly critical when the computer isn't working as it should be—for instance, if it's unusually slow. The ps program supports an unusual number of options, but just a few of them will take you a long way. Likewise, interpreting ps output can be tricky because so many options modify the program's output. Some ps-like programs, most notably top, also deserve some attention.

Useful *ps* Options

The official syntax for ps is fairly simple:

```
ps [options]
```

This simplicity of form hides considerable complexity because ps supports three different *types* of options, as well as many options within each type. The three types of options are as follows:

Unix98 options   These single-character options may be grouped together and are preceded by a single dash (-).

BSD options   These single-character options may be grouped together and must *not* be preceded by a dash.

GNU long options   These multi-character options are never grouped together. They're preceded by two dashes (--).

Options that may be grouped together may be clustered without spaces between them. For instance, rather than typing **ps -a -f**, you can type **ps -af**. The reason for so much complexity is that the ps utility has historically varied a lot from one Unix OS to another. The version of ps that ships with major Linux distributions attempts to implement most features from all these different ps versions, so it supports many different personalities. In fact, you can change some of its default behaviors by setting the PS_PERSONALITY environment variable to posix, old, linux, bsd, sun, digital, or various others. The rest of this section describes the default ps behavior on most Linux systems.

Some of the more useful ps features include the following:

Display help   The --help option presents a summary of some of the more common ps options.

Display all processes   By default, ps displays only processes that were run from its own terminal (xterm, text-mode login, or remote login). The -A and -e options cause it to display all the processes on the system, and x displays all processes owned by the user who gives the command. The x option also increases the amount of information that's displayed about each process.

Display one user's processes   You can display processes owned by a given user with the -u *user*, U *user*, and --User *user* options. The *user* variable may be a username or a user ID.

Display extra information   The -f, -l, j, l, u, and v options all expand the information provided in the ps output. Most ps output formats include one line per process, but ps can display enough information that it's impossible to fit it all on one 80-character line. Therefore, these options provide various mixes of information.

Display process hierarchy   The -H, -f, and --forest options group processes and use indentation to show the hierarchy of relationships between processes. These options are useful if you're trying to trace the parentage of a process.

Display wide output   The ps command output can be more than 80 columns wide. Normally, ps truncates its output so that it will fit on your screen or xterm. The -w and w options tell ps not to do this, which can be useful if you direct the output to a file, as **in ps w > ps.txt**. You can then examine the output file in a text editor that supports wide lines.

You can combine these ps options in many ways to produce the output you want. You'll probably need to experiment to learn which options produce the desired results because each of these options modifies the output in some way. Even those that would seem to influence just the selection of processes to list sometimes modify the information that's provided about each process.

Interpreting *ps* Output

Listings 1.4 and 1.5 show a couple of examples of ps in action. Listing 1.4 shows **ps -u rodsmith --forest**, and Listing 1.5 shows **ps u U rodsmith**.

Listing 1.4: Output of *ps -u rodsmith --forest*

```
$ ps -u rodsmith --forest
  PID TTY          TIME CMD
 2451 pts/3    00:00:00 bash
 2551 pts/3    00:00:00 ps
 2496 ?        00:00:00 kvt
 2498 pts/1    00:00:00 bash
 2505 pts/1    00:00:00  \_ nedit
 2506 ?        00:00:00       \_ csh
 2544 ?        00:00:00           \_ xeyes
19221 ?        00:00:01 dfm
```

Listing 1.5: Output of *ps u U rodsmith*

```
$ ps u U rodsmith
USER        PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
rodsmith 19221  0.0  1.5  4484 1984 ?        S    May07   0:01 dfm
rodsmith  2451  0.0  0.8  1856 1048 pts/3    S    16:13   0:00 -bash
rodsmith  2496  0.2  3.2  6232 4124 ?        S    16:17   0:00 /opt/kd
rodsmith  2498  0.0  0.8  1860 1044 pts/1    S    16:17   0:00 bash
rodsmith  2505  0.1  2.6  4784 3332 pts/1    S    16:17   0:00 nedit
```

```
rodsmith  2506  0.0  0.7  2124 1012 ?       S    16:17    0:00 /bin/cs
rodsmith  2544  0.0  1.0  2576 1360 ?       S    16:17    0:00 xeyes
rodsmith  2556  0.0  0.7  2588  916 pts/3   R    16:18    0:00 ps u U
```

The output produced by ps normally begins with a heading line, which displays the meaning of each column. Important information that might be displayed (and labeled) includes the following:

Username   The name of the user who runs the programs. Listings 1.4 and 1.5 restricted this output to one user to limit the size of the listings.

Process ID   The process ID (PID) is a number that's associated with the process. This item is particularly important because you need it to modify or kill the process, as described later in this chapter.

Parent process ID   The parent process ID (PPID) identifies the process's parent. (Neither Listing 1.4 nor Listing 1.5 shows the PPID, though.)

TTY   The teletype (TTY) is a code used to identify a terminal. As illustrated by Listings 1.4 and 1.5, not all processes have TTY numbers—X programs and daemons, for instance, do not. Text-mode programs do have these numbers, though, which point to a console, xterm, or remote login session.

CPU time   The TIME and %CPU headings are two measures of CPU time used. The first indicates the total amount of CPU time consumed, and the second represents the percentage of CPU time the process is using when ps executes. Both can help you spot runaway processes—those that are consuming too much CPU time. Unfortunately, just what constitutes "too much" varies from one program to another, so it's impossible to give a simple rule to help you spot a runaway process.

CPU priority   As described shortly, in "Managing Process Priorities," it's possible to give different processes different priorities for CPU time. The NI column, if present (it's not in the preceding examples) lists these priority codes. The default value is 0. Positive values represent *reduced* priority, while negative values represent *increased* priority.

Memory use   Various headings indicate memory use—for instance, RSS is resident set size (the memory used by the program and its data) and %MEM is the percentage of memory the program is using. Some output formats also include a SHARE column, which is memory that's shared with other processes (such as shared libraries). As with CPU use measures, these columns can help point you to the sources of difficulties, but because legitimate memory needs of programs vary so much, it's impossible to give a simple criterion for when a problem exists.

Command   The final column in most listings is the command used to launch the process. This is truncated in Listing 1.5 because this format lists the complete command, but so much other information appears that the complete command won't usually fit on one line. (This is where the wide-column options can come in handy.)

As you can see, a lot of information can be gleaned from a ps listing—or perhaps that should be the plural *listings*, because no one format includes all of the available information. For the most part, the PID, username, and command are the most important pieces of information. In some

cases, though, you may need specific other components. If your system's memory or CPU use has skyrocketed, for instance, you'll want to pay attention to the memory or CPU use columns.

> It's often necessary to find specific processes. You might want to find the PID associated with a particular command in order to kill it, for instance. This information can be gleaned by piping the ps output through grep, as in **ps ax | grep bash** to find all the instances of bash.

Although you may need a wide screen or xterm to view the output, you may find **ps -A --forest** to be a helpful command in learning about your system. Processes that don't fall off others were either started directly by init or have had their parents killed, and so they have been "adopted" by init. (Chapter 6 describes init and the boot procedure in more detail.) Most of these processes are fairly important—they're servers, login tools, and so on. Processes that hang off several others in this tree view, such as xeyes and nedit in Listing 1.4, are mostly user programs launched from shells.

top: A Dynamic ps Variant

If you want to know how much CPU time various processes are consuming relative to one another, or if you simply want to quickly discover which processes are consuming the most CPU time, a tool called top is the one for the job. The top tool is a text-mode program, but of course it can be run in an xterm or similar window, as shown in Figure 1.2, and there are also GUI variants, like kpm and gnome-system-monitor. By default, top sorts its entries by CPU use, and it updates its display every few seconds. This makes it a very good tool for spotting runaway processes on an otherwise lightly loaded system—those processes almost always appear in the first position or two, and they consume an inordinate amount of CPU time. By looking at Figure 1.2, you might think that FahCore_65.exe is such a process, but in fact, it's legitimately consuming a lot of CPU time. You'll need to be familiar with the purposes and normal habits of programs running on *your* system in order to make such determinations; the legitimate needs of different programs vary so much that it's impossible to give a simple rule for judging when a process is consuming too much CPU time.

Like many Linux commands, top accepts several options. The most useful of these options are listed here:

-d *delay*   This specifies the delay between updates, which is normally 5 seconds.

-p *pid*   If you want to monitor specific processes, you can list them using this option. You'll need the PIDs, which you can obtain with ps, as described earlier. You can specify up to 20 PIDs by using this option multiple times, once for each PID.

-n *iter*   You can tell top to display a certain number of updates (*iter*) and then quit. (Normally, top continues updating until you terminate the program.)

-b   This specifies batch mode, in which top doesn't use the normal screen update commands. You might use this to log CPU use of targeted programs to a file, for instance.

FIGURE 1.2 The top command shows system summary information and information on the most CPU-intensive processes on a computer.


```
  ─ ○ ⍓                          Shell - Konsole                        ▫ □ ✕

  Session  Edit  View  Bookmarks  Settings  Help

  top - 21:51:03 up 20 days, 11:25, 20 users,  load average: 2.09, 2.10, 2.02
  Tasks: 150 total,   3 running, 146 sleeping,   0 stopped,   1 zombie
  Cpu(s):  2.0% us,  0.0% sy, 98.0% ni,  0.0% id,  0.0% wa,  0.0% hi,  0.0% si
  Mem:    513056k total,   509992k used,     3064k free,    36588k buffers
  Swap:   683888k total,   237528k used,   446360k free,   140308k cached

    PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
   8673 root      39  19 70648   9m 2112 R 98.2  2.0  1664:29 FahCore_65.exe
   8774 rodsmith  15   0 48080 4992   45m S  1.3  1.0 411:12.66 artsd
  12292 root      15   0  156m  67m  66m S  0.3 13.4 175:55.95 X
      1 root      16   0  2476  480 2312 S  0.0  0.1   0:02.39 init
      2 root      34  19     0    0    0 S  0.0  0.0   0:00.11 ksoftirqd/0
      3 root       5 -10     0    0    0 S  0.0  0.0   0:05.84 events/0
      4 root       5 -10     0    0    0 S  0.0  0.0   0:42.30 kblockd/0
      6 root       6 -10     0    0    0 S  0.0  0.0   0:00.00 khelper
      5 root      15   0     0    0    0 S  0.0  0.0   0:00.11 khubd
     10 root      15 -10     0    0    0 S  0.0  0.0   0:00.00 aio/0
      9 root      15   0     0    0    0 S  0.0  0.0   0:18.85 kswapd0
     99 root      25   0     0    0    0 S  0.0  0.0   0:00.00 scsi_eh_0
    104 root      15   0     0    0    0 S  0.0  0.0   0:00.00 scsi_eh_1
    106 root      16   0     0    0    0 S  0.0  0.0   0:00.00 scsi_eh_2
    107 root      15   0     0    0    0 S  0.0  0.0   0:00.00 katad-2
    137 root      16   0     0    0    0 S  0.0  0.0   0:00.00 kseriod
    178 root       5 -10     0    0    0 S  0.0  0.0   0:07.52 reiserfs/0

  ▣ ▣ Shell   ▣ Shell No. 2   ▣ Shell No. 3
```

You can do more with top than watch it update its display. When it's running, you can enter any of several single-letter commands, some of which prompt you for additional information. These commands include the following:

h or ?   These keystrokes display help information

k   You can kill a process with this command. The top program will ask for a PID number, and if it's able to kill the process, it will do so. (The upcoming section "Killing Processes" describes other ways to kill processes.)

q   This option quits from top.

r   You can change a process's priority with this command. You'll have to enter the PID number and a new priority value—a positive value will decrease its priority and a negative value will increase its priority, assuming it has the default 0 priority to begin with. Only root may increase a process's priority. The renice command (described shortly, in "Managing Process Priorities") is another way to accomplish this task.

s   This command changes the display's update rate, which you'll be asked to enter (in seconds).

P   This sets the display to sort by CPU usage, which is the default.

M   You can change the display to sort by memory usage with this command.

More commands are available in top (both command-line options and interactive commands) than can be summarized here; consult top's man page for more information.

One of the pieces of information provided by top is the *load average*, which is a measure of the demand for CPU time by applications. In Figure 1.2, you'll see three load-average estimates on the top line; these correspond to the current load average and two previous

measures. A system on which no programs are demanding CPU time will have a load average of 0. A system with one program running CPU-intensive tasks will have a load average of 1. Higher load averages reflect programs competing for available CPU time. You can also find the current load average via the `uptime` command, which displays the load average along with information on how long the computer has been running. The load average can be useful in detecting runaway processes. For instance, if a system normally has a load average of 0.5 but it suddenly gets stuck at a load average of 2.5, there may be a couple of CPU-hogging processes that have *hung*—that is, become unresponsive. Hung processes sometimes needlessly consume a lot of CPU time. You can use `top` to locate these processes and, if necessary, kill them.

jobs: Processes Associated with Your Session

The `jobs` command displays minimal information on the processes associated with the current session. In practice, `jobs` is usually of limited value, but it does have a few uses. One of these is to provide job ID numbers. These numbers are conceptually similar to PID numbers, but they're not the same. Jobs are numbered starting from 1 for each session, and in most cases, a single shell has only a few associated jobs. The job ID numbers are used by a handful of utilities in place of PIDs, so you may need this information.

A second use of `jobs` is to ensure that all your programs have terminated prior to logging out. Under some circumstances, logging out of a remote login session can cause the client program to freeze up if you've left programs running. A quick check with `jobs` will inform you of any forgotten processes and enable you to shut them down.

Foreground and Background Processes

One of the most basic process management tasks is to control whether a process is running in the foreground or the background—that is, whether or not it's monopolizing the use of the terminal from which it was launched. Normally, when you launch a program it takes over the terminal, preventing you from doing other work in that terminal. (Some programs, though, release the terminal. This is most common for servers and some GUI programs.)

If a program is running but you decide you want to use that terminal for something else, pressing Ctrl+Z normally pauses the program and gives you control of the terminal. (An important point is that this procedure suspends the program, so if it's performing real work, that work stops!) This can be handy if, say, you're running a text editor in a text-mode login and you want to check a filename so you can mention it in the file you're editing. You'd press Ctrl+Z and type `ls` to get the file listing. To get back to the text editor, you'd then type `fg`, which restores the text editor to the foreground of your terminal. If you've suspended several processes, you'd add a job number, as in `fg 2` to restore job 2. You can obtain a list of jobs associated with a terminal by typing `jobs`, which displays the jobs and their job numbers.

A variant on `fg` is `bg`. Where `fg` restores a job to the foreground, `bg` restores a job to running status, but in the background. You might use this command if the process you're running is performing a CPU-intensive task that requires no human interaction but you want to use the terminal in the meantime. Another use of `bg` is in a GUI environment—after launching a GUI

program from an xterm or similar window, that shell is tied up servicing the GUI program, which probably doesn't really need the shell. Pressing Ctrl+Z in the xterm window will enable you to type shell commands again, but the GUI program will be frozen. To unfreeze the GUI program, type **bg** in the shell, which enables the GUI program to run in the background while the shell continues to process your commands.

An alternative to launching a program, using Ctrl+Z, and typing **bg** to run a program in the background is to append an ampersand (&) to the command when launching the program. For instance, rather than edit a file with the NEdit GUI editor by typing **nedit myfile.txt**, you could type **nedit myfile.txt &**. This command launches the nedit program in the background from the start, leaving you able to control your xterm window for other tasks.

Managing Process Priorities

There may be times when you'll want to prioritize your programs' CPU use. For instance, you might be running a program that's very CPU-intensive but that will take a long time to finish its work, and you don't want that program to interfere with others that are of a more interactive nature. Alternatively, on a heavily loaded computer, you might have a job that's more important than others that are running, so you might want to give it a priority boost. In either case, the usual method of accomplishing this goal is through the nice and renice commands. You can use nice to launch a program with a specified priority or use renice to alter the priority of a running program.

You can assign a priority to nice in any of three ways: by specifying the priority preceded by a dash (this works well for positive priorities but makes them look like negative priorities), by specifying the priority after a -n parameter, or by specifying the priority after an --adjustment= parameter. In all cases, these parameters are followed by the name of the program you want to run:

```
nice [argument] [command [command-arguments]]
```

For instance, the following three commands are all equivalent:

```
$ nice -12 number-crunch data.txt
$ nice -n 12 number-crunch data.txt
$ nice --adjustment=12 number-crunch data.txt
```

All three of these commands run the number-crunch program at priority 12 and pass it the data.txt file. If you omit the adjustment value, nice uses 10 as a default. The range of possible values is –20 to 19, with negative values having the highest priority. Only root may launch a program with increased priority (that is, give a negative priority value), but any user may use nice to launch a program with low priority. The default priority for a program run without nice is 0.

If you've found that a running process is consuming too much CPU time or is being swamped by other programs and so should be given more CPU time, you can use the renice program to alter its priority without disrupting the program's operation. The syntax for renice is as follows:

```
renice priority [[-p] pids] [[-g] pgrps] [[-u] users]
```

You must specify the *priority*, which takes the same values this variable takes with `nice`. In addition, you must specify one or more PIDs (*pids*), one or more group IDs (*pgrps*), or one or more usernames (*users*). In the latter two cases, `renice` changes the priority of all programs that match the specified criterion—but only `root` may use `renice` in this way. Also, only `root` may increase a process's priority. If you give a numeric value without a `-p`, `-g`, or `-u` option, `renice` assumes the value is a PID. You may mix and match these methods of specification. For instance, you might enter the following command:

```
# renice 7 16580 -u pdavison tbaker
```

This command sets the priority to 7 for PID 16580 and for all processes owned by `pdavison` and `tbaker`.

Killing Processes

Sometimes reducing a process's priority isn't a strong enough action. A program may have become totally unresponsive, or you might want to terminate a process that shouldn't be running at all. In these cases, the `kill` command is the tool to use. This program sends a *signal* (a method that Linux uses to communicate with processes) to a process. The signal is usually sent by the kernel, the user, or the program itself to terminate the process. Linux supports many numbered signals, each of which is associated with a specific name. You can see them all by typing **kill -l**. If you don't use `-l`, the syntax for `kill` is as follows:

```
kill -s signal pid
```

> Although Linux includes a `kill` program, many shells, including bash and `csh`, include built-in `kill` equivalents that work in much the same way as the external program. If you want to be sure you're using the external program, type its complete path, as in `/bin/kill`.

The `-s signal` parameter sends the specified signal to the process. You can specify the signal using either a number (such as 9) or a name (such as `SIGKILL`). The signals you're most likely to use are 1 (`SIGHUP`, which terminates interactive programs and causes many daemons to reread their configuration files), 9 (`SIGKILL`, which causes the process to exit without performing routine shutdown tasks), and 15 (`SIGTERM`, which causes the process to exit but allows it to close open files and so on). If you don't specify a signal, the default is 15 (`SIGTERM`). You can also use the shortened form `-signal`. If you do this and use a signal name, you should omit the `SIG` portion of the name—for instance, use `KILL` rather than `SIGKILL`. The *pid* option is, of course, the PID for the process you want to kill. You can obtain this number from `ps` or `top`.

> The `kill` program will only kill processes owned by the user who runs `kill`. The exception is if that user is `root`; the superuser may kill any user's processes.

Real World Scenario

Running Programs Persistently

Signals can be passed to programs by the kernel even if you don't use the kill command. For instance, when you log out of a session, the programs you started from that session are sent the SIGHUP signal, which causes them to terminate. If you want to run a program that will continue running even when you log out, you can launch it with the nohup program:

```
$ nohup program options
```

This command causes the program to ignore the SIGHUP signal. It could be handy if you want to launch certain small servers that may legitimately be run as ordinary users.

A variant on kill is killall, which has the following form:

```
killall [options] [--] name [...]
```

This command kills a process based on its name rather than its PID number. For instance, **killall vi** kills all the running processes called vi. You may specify a signal in the shortened form (-*signal*) or by preceding the signal number with -s or --signal. As with kill, the default is 15 (SIGTERM). One potentially important option to killall is -i, which causes it to ask for confirmation before sending the signal to each process. You might use it like this:

```
$ killall -i vi
Kill vi(13211) ? (y/n) y
Kill vi(13217) ? (y/n) n
```

In this example, two instances of the Vi editor were running but only one should have been killed. As a general rule, if you run killall as root, you should use the -i parameter; if you don't, it's all too likely that you'll kill processes that you should not, particularly if the computer is being used by many people at once.

> **WARNING**   Some versions of Unix provide a killall command that works very differently from Linux's killall. This alternate killall kills all the processes started by the user who runs the command. This is a potentially much more destructive command, so if you ever find yourself on a non-Linux system, *do not* use killall until you've discovered what that system's killall does, say by reading the killall man page.

Summary

The command line is the key to Linux. Even if you prefer GUI tools to text-mode tools, understanding text-mode commands is necessary to fully manage Linux. This task begins with the shell, which accepts commands you type and displays the results of those commands. In addition, shells support linking programs together via pipes and redirecting programs' input and output. These features enable you to perform complex tasks using simple tools by having each program perform its own small part of the task. This technique is frequently used with Linux text filters, which manipulate text files in various ways—sorting text by fields, merging multiple files, and so on. Beyond text filters, Linux also supports many text editors, but the most basic Linux text editor is Vi. Although it's very strange by modern standards, Vi is a compact and efficient editor that's found on just about every Linux emergency system ever built, so knowing how to use it can help you when you need to use such a system.

Beyond manipulating text files, using Linux involves manipulating processes. Knowing how to manage foreground and background processes, adjust process priorities, and kill stray processes can help you keep your Linux system working well.

Exam Essentials

Summarize features that Linux shells offer to speed up command entry.   The command history often enables you to retrieve an earlier command that's similar or identical to the one you want to enter. Tab completion reduces typing effort by letting the shell finish long command names or filenames. Command line editing enables you to edit a retrieved command or change a typo before committing the command.

Describe the purpose of the `man` command.   The `man` command displays the manual page for the keyword (command, filename, system call, or other feature) that you type. This documentation provides succinct summary information that's useful as a reference to learn about exact command options or features.

Explain the purpose of environment variables.   Environment variables store small pieces of data—program options, information about the computer, and so on. This information can be read by programs and used to modify program behavior in a way that's appropriate for the current environment.

Describe the difference between standard output and standard error.   Standard output carries normal program output, whereas standard error carries high-priority output, such as error messages. The two can be redirected independently of one another.

Explain the purpose of pipes.   Pipes tie programs together by feeding the standard output from the first program into the second program's standard input. They can be used to link together a series of simple programs to perform more complex tasks than any one of the programs could manage.

Summarize the structure of regular expressions.   Regular expressions are strings that describe other strings. They contain normal alphanumeric characters, which match the exact same characters, as well as several special symbols and symbol sets that can be used to match multiple different characters. The combination is a powerful pattern-matching tool used by many Linux programs.

Describe Vi's three editing modes.   You enter text using the insert mode, which supports text entry and deletion. The command and ex modes are used to perform more complex commands or run outside programs to operate on the text entered or changed in insert mode.

Explain the difference between foreground and background processes.   Foreground processes have control of the current terminal or text-mode window (such as an `xterm`). Background processes do not have exclusive control of a terminal or text-mode window but are still running.

Describe how to limit the CPU time used by a process.   You can launch a program with `nice` or use `renice` to alter its priority in obtaining CPU time. If a process is truly out of control, you can terminate it with the `kill` command.

Review Questions

1. You type a command into `bash` and pass a long filename to it, but after you enter the command, you receive a `File not found` error message because of a typo in the filename. How might you proceed?

    A. Retype the command and be sure you type the filename correctly, letter by letter.

    B. Retype the command, but press the Tab key after typing a few letters of the long filename to ensure that the filename is entered correctly.

    C. Press the up arrow key and use `bash`'s editing features to correct the typo.

    D. Any of the above.

2. Which of the following commands is implemented as an internal command in `bash`?

    A. `cat`

    B. `echo`

    C. `tee`

    D. `sed`

3. You type **`echo $PROC`** and the computer replies `Go away`. What does this mean?

    A. No currently running processes are associated with your shell, so you may log out without terminating them.

    B. The remote computer PROC is not accepting connections; you should contact its administrator to correct the problem.

    C. Your computer is handling too many processes; you must kill some of them to regain control of the computer.

    D. You, one of your configuration files, or a program you've run has set the $PROC environment variable to `Go away`.

4. How does `man` display information by default on most Linux systems?

    A. Using a custom X-based application

    B. Using the Mozilla web browser

    C. Using the `less` pager

    D. Using the Vi editor

5. You want to store the standard output of the `ifconfig` command in a text file (`file.txt`) for future reference, and you want to wipe out any existing data in the file. How would you do so?

    A. `ifconfig < file.txt`

    B. `ifconfig >> file.txt`

    C. `ifconfig > file.txt`

    D. `ifconfig | file.txt`

6. What is the effect of the following command?

```
$ myprog &> input.txt
```

   A. Standard error to `myprog` is taken from `input.txt`.

   B. Standard input to `myprog` is taken from `input.txt`.

   C. Standard input and standard error from `myprog` are written to `input.txt`.

   D. All of the above.

7. How many commands can you pipe together at once?

   A. 2

   B. 3

   C. 4

   D. An arbitrary number

8. What program would you use to display the end of a configuration file?

   A. `uniq`

   B. `cut`

   C. `tail`

   D. `wc`

9. What is the effect of the following command?

```
$ pr report.txt | lpr
```

   A. The file `report.txt` is formatted for printing and sent to the `lpr` program.

   B. The files `report.txt` and `lpr` are combined together into one file and sent to standard output.

   C. Tabs are converted to spaces in `report.txt` and the result is saved in `lpr`.

   D. None of the above.

10. Which of the following commands will number the lines in `aleph.txt`? (Select all that apply.)

   A. `fmt aleph.txt`

   B. `nl aleph.txt`

   C. `cat -b aleph.txt`

   D. `cat -n aleph.txt`

11. Which of the following commands will change all occurrences of `dog` in the file `animals.txt` to `mutt` in the screen display?

   A. `sed −s "dog" "mutt" animals.txt`

   B. `grep −s "dog||mutt" animals.txt`

   C. `sed 's/dog/mutt/' animals.txt`

   D. `cat animals.txt | grep −c "dog" "mutt"`

12. Which of the following commands will print lines from the file `world.txt` that contain matches to `changes` and `changed`?

    A. `grep change[ds] world.txt`

    B. `sed change[d-s] world.txt`

    C. `od "change'd|s'" world.txt`

    D. `cat world.txt changes changed`

13. Which of the following regular expressions will match the strings `dig` and `dug` but not `dog`?

    A. `d.g`

    B. `d[iu]g`

    C. `d[i-u]g`

    D. `di*g`

14. How would you remove two lines of text from a file using Vi?

    A. In command mode, position the cursor on the first line and type **2dd**.

    B. In command mode, position the cursor on the last line and type **2yy**.

    C. In insert mode, position the cursor at the start of the first line, hold the Shift key down while pressing the down arrow key twice, and hit the Delete key on the keyboard.

    D. In insert mode, position the cursor at the start of the first line and press Ctrl+K twice.

15. A user types `kill -9 11287` at a `bash` prompt. What is the probable intent, assuming the user typed the correct command?

    A. To cut off a network connection using TCP port 11287

    B. To display the number of processes that have been killed with signal 11287 in the last nine days

    C. To cause a server with process ID 11287 to reload its configuration file

    D. To terminate a misbehaving or hung program with process ID 11287

16. What programs might you use to learn what your system's load average is? (Select all that apply.)

    A. `ld`

    B. `load`

    C. `top`

    D. `uptime`

17. Which of the following commands creates a display of processes, showing the parent/child relationships through links between their names?

    A. `ps --forest`

    B. `ps aux`

    C. `ps -e`

    D. All of the above

18. You use top to examine the CPU time being consumed by various processes on your system. You discover that one process, dfcomp, is consuming over 90 percent of your system's CPU time. What can you conclude?

    A. Very little; dfcomp could be legitimately consuming that much CPU time or it could be an unauthorized or malfunctioning program.

    B. No program should consume 90 percent of available CPU time; dfcomp is clearly malfunctioning and should be terminated.

    C. This is normal; dfcomp is the kernel's main scheduling process, and it consumes any unused CPU time.

    D. This behavior is normal *if* your CPU is less powerful than a 1.5GHz Pentium 4, but on newer systems, no program should consume 90 percent of CPU time.

19. Which two of the following commands are equivalent to one another? (Select two.)

    A. `nice --value 10 crunch`

    B. `nice -n -10 crunch`

    C. `nice -10 crunch`

    D. `nice crunch`

20. Which of the following are restrictions on ordinary users' abilities to run renice? (Select all that apply.)

    A. Users may not modify the priorities of processes that are already running.

    B. Users may not modify the priorities of other users' processes.

    C. Users may not decrease the priority (that is, increase the priority value) of their own processes.

    D. Users may not increase the priority (that is, decrease the priority value) of their own processes.

Answers to Review Questions

1. D. Any of these approaches will work, or at least *could* work. (You might err when performing any of them.) Option B or C is likely to be the most efficient approach, though; with a long file-name to type, option A is likely to be tedious.

2. B. The `echo` command is implemented internally to `bash`, although an external version is also available on most systems. The `cat`, `tee`, and `sed` commands are not implemented internally to `bash`, although they can be called from `bash` as external commands.

3. D. The `echo` command echoes what follows to standard output, and $PROC is an environment variable. Thus, **echo $PROC** displays the value of the $PROC environment variable, meaning that it must have been set to the specified value by you, one of your configuration files, or a program you've run. Although many environment variables are set to particular values to convey information, $PROC is not a standard environment variable that might be associated with information described in options A, B, or C.

4. C. By default, `man` uses the `less` pager to display information on most Linux systems. Although an X-based version of `man` does exist (`xman`), the basic `man` doesn't use a custom X-based application, nor does it use Mozilla or the Vi editor.

5. C. The > redirection operator stores a command's standard output in a file, overwriting the contents of any existing file by the specified name. Option A specifies the standard input redirection so that `ifconfig` will take the contents of `file.txt` as input. Option B is almost correct; the >> redirection operator redirects standard output, as requested, but it appends data to the specified file rather than overwriting it. Option D specifies a pipe; the output of `ifconfig` is sent through the `file.txt` program, if it exists. (Chances are it doesn't, so you'd get a `command not found` error message.)

6. C. The &> redirection operator sends both standard output and standard error to the specified file, as option C states. (The name of the file, `input.txt`, is a bit of misdirection, but the usage is still valid.) Option A mentions standard error but describes it as if it were an input stream, which it's not; it's an output stream. Option B mentions standard input, but the &> operator doesn't affect standard input.

7. D. In principle, you could pipe together as many commands as you like. (In practice, of course, there will be limits based on input buffer size, memory, and so on, but these limits are far higher than the 2, 3, or 4 commands specified in options A, B, and C.)

8. C. The `tail` command displays the final 10 lines of a file. (You can change the number of lines displayed with the `-n` option.) The `uniq` command removes duplicate lines from a list. The `cut` command echoes the specified characters or fields from an input text file. The `wc` command displays counts of the number of characters, words, and lines in a file.

9. A. The `pr` program takes a text file as input and adds formatting features intended for printing, such as a header and blank lines to separate pages. The command also pipes the output through `lpr` (which is a Linux printing command).

10. B, C, D. The `nl` command numbers lines, so it does this task without any special options. (Its options can fine-tune the way it numbers lines, though.) The `cat` command can also number lines via its `-b` or `-n` options; `-b` numbers non-blank lines, while `-n` numbers all lines (including blank lines). The `fmt` command is the only one described here that will not number the lines of the input file.

11. C. The `sed` utility can be used to "stream" text and change one value to another. In this case, the `s` option is used to replace `dog` with `mutt`. The syntax in option A is incorrect, while choices B and D are incorrect since `grep` does not include the functionality needed to make the changes.

12. A. The `grep` utility is used to find matching text within a file and print those lines. It accepts regular expressions, which allow for the placing of the two characters you are looking for within brackets. The syntax for `sed`, `od`, and `cat` would not perform the specified task.

13. B. The bracket expression within the `d[iu]g` regular expression means that either `i` or `u` may be the middle character; hence, this regular expression matches both `dig` and `dug` but not `dog`. Option A's dot matches any single character, so `d.g` matches all three words. The range expression `[i-u]` matches any single character between `i` and `u`, inclusive. Because `o` falls between these two letters, option C matches all three words. Finally, `di*g` matches `dig`, `diig`, `diiig`, or any other word that begins with `d`, ends with `g` and contains any number of `i` letters in-between. Thus, option D matches `dig` but not `dug` as required.

14. A. In Vi, `dd` is the command-mode command that deletes lines. Preceding this command by a number deletes that number of lines. While `yy` works similarly, it copies ("yanks") text rather than deleting it. Option C works in many more modern text editors, but not in Vi. Option D works in Emacs and similar text editors, but not in Vi.

15. D. The `kill` program accepts various signals in numeric or named form (`9` in this example) along with a process ID number (`11287` in this example). Signal 9 corresponds to `SIGKILL`, which is an extreme way to kill processes that have run out of control. Although you might use `kill` to kill network processes, you can't pass `kill` a TCP port number and expect it to work. The program also won't display information on the number of processes that have been killed. To do as option C suggests, you'd need to tell `kill` to pass `SIGHUP` (signal 1), so the command would be **`kill -1 11287`**.

16. C, D. The `top` utility displays a dynamic list of processes ordered according to their CPU use along with additional system information, including load averages. If you want only the load average at a specific moment, `uptime` may be better because it presents less extraneous information—it shows the current time, the time since the system was booted, the number of active users, and the load averages. The `ld` command has nothing to do with displaying load averages (it's a programming tool that links together program modules into an executable program). There is no standard Linux program called `load`.

17. A. The `--forest` option to `ps` shows parent/child relationships by creating visual links between process names in the `ps` output. (Listing 1.4 shows this effect.) Options B and C are both valid `ps` commands, but neither creates the specified effect.

18. A. CPU-intensive programs routinely consume 90 percent or more of available CPU time, but not all systems run such programs. Furthermore, some types of program bugs can create such CPU loads. Thus, you must investigate the matter more. What is `dfcomp`? Is it designed as a CPU-intensive program? Is it consuming this much CPU time consistently, or was this a brief burst of activity?

19. Answers: C, D. The `nice` command launches a program (`crunch` in this example) with increased or decreased priority. The default priority when none is specified is 10, and the **`nice -10 crunch`** command also sets the priority to 10, so options C and D are equivalent. Option A isn't a valid `nice` command because `nice` has no `--value` option. Option B is a valid `nice` command, but it sets the priority to –10 rather than 10.

20. Answers: B, D. Linux insulates users' actions from one another, and this rule applies to `renice`; only `root` may modify the priority of other users' processes. Similarly, only `root` may increase the priority of a process, in order to prevent users from setting their processes to maximum priority, thus stealing CPU time from others. Option A correctly describes `nice`, but not `renice`; the whole point of `renice` is to be able to change the priorities of existing processes. Option C also describes an action that `renice` permits.

Chapter

2

Managing Software

THE FOLLOWING LINUX PROFESSIONAL INSTITUTE OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **1.102.3 Make and install programs from source (weight: 5)**
- ✓ **1.102.4 Manage shared libraries (weight: 4)**
- ✓ **1.102.5 Use Debian package management (weight: 8)**
- ✓ **1.102.6 Use Red Hat Package Manager (RPM) (weight: 8)**

A Linux system is defined largely by the collection of software it contains. The Linux kernel, libraries used by many packages, the shells used to interpret commands, the X Window System GUI, servers, and more all make up the system's software environment. Many of the chapters of this book are devoted to configuring specific software components, but they all have something in common: tools used to install, uninstall, upgrade, and otherwise manipulate the software itself. Ironically, this commonality is a major source of differences between Linux systems. Two major Linux package management tools exist: the *RPM Package Manager (RPM)* and *Debian packages.* (Several less-common package management systems also exist.) With few exceptions, each individual Linux computer uses precisely one package management system, so you'll need to know only one to administer a single system. To be truly fluent in all things Linux, though, you should be at least somewhat familiar with both of them. Thus, this chapter describes both.

This chapter also covers another common method of installing software: building it from source code. Most Linux software is available in source code form, and some administrators like installing as much software as possible from source code because doing so enables them to tweak compiler settings and be sure that no strange patches have been applied to the software. Even if you prefer the convenience of binary package installation via RPM or Debian packages, you should be familiar with the basic procedures for installing software from source code so that you can do so if you must—say, if you can't find an obscure but important (for you) package in binary form for your system. Finally, this chapter covers *libraries*—software components that can be used by many different programs. Libraries help reduce the disk space and memory requirements of complex programs, but they also require some attention, and if that attention isn't given to them, they can cause problems by their absence or because of incompatibilities between their and their dependent software's versions.

Package Concepts

Before proceeding further, you should understand some of the principles that underlie Linux package management tools. Any computer's software is like a house of cards: One program may rely upon five other programs or libraries, each of which rely upon several more, and so on. The foundation upon which all these programs rely is the Linux kernel. Any of these packages can theoretically be replaced by an equivalent one; however, doing so sometimes causes problems. Worse, removing one card from the stack could cause the whole house of cards to come tumbling down.

Linux package management tools are intended to minimize such problems by tracking what software is installed. The information that the system maintains helps avoid problems in several ways:

Packages   The most basic information that package systems maintain is information on software *packages*—that is, collections of files that are installed on the computer. Packages are usually distributed as single files that are similar to *tarballs* (archives created with the `tar` utility and usually compressed with `gzip` or `bzip2`) or zip files. Once installed, most packages consist of dozens or hundreds of files, and the package system tracks them all. Packages include additional information that aids in the subsequent duties of package management systems, though.

Installed file database   Package systems maintain a database of installed files. The database includes information on every file installed via the package system, the name of the package to which it belongs, and associated additional information.

Dependencies   One of the most important pieces of information maintained by the package system is *dependency* information—that is, the requirements of packages for one another. For instance, if SuperProg relies on UltraLib to do its work, the package database records this information. If you attempt to install SuperProg when UltraLib isn't installed, the package system won't let you do so. Similarly, if you try to uninstall UltraLib when SuperProg is installed, the package system won't let you. (You can override these prohibitions, as described later, in "Forcing the Installation." Doing so is usually inadvisable, though.)

Checksums   The package system maintains checksums and assorted ancillary information about files. This information can be used to verify the validity of the installed software. This feature has its limits, though; it's intended to help you spot disk errors, accidental overwriting of files, or other non-sinister problems. It's of limited use in detecting intrusions because an intruder could use the package system to install altered system software.

Upgrades and uninstallation   By tracking files and dependencies, package systems permit easy upgrades and uninstallation: Tell the package system to upgrade or remove a package and it will replace or remove every file in the package. Of course, this assumes that the upgrade or uninstallation doesn't cause dependency problems; if it does, the package system will block the operation unless you override it.

Binary package creation   Both RPM and Debian package systems provide tools to help create binary packages (those that are installed directly) from source code. This feature is particularly helpful if you're running Linux on a peculiar CPU; you can download source code and create a binary package even if the developers didn't provide explicit support for your CPU. Creating a binary package from source has advantages over compiling software from source in more conventional ways, as described in "Installing Programs from Source," because you can then use the package management system to track dependencies, attend to individual files, and so on.

Both the RPM and Debian package systems provide all of these basic features, although the details of their operation differ. These two package systems are incompatible with one another in the sense that their package files and their installed file databases are different; you can't directly install an RPM package on a Debian-based system or vice versa. (Tools to convert between formats do exist, though, and developers are working on ways to better integrate the two package formats.)

WARNING   Most distributions install just one package system. It's possible to install more than one, though, and in fact some programs (such as `alien`) require both for full functionality. Actually *using* both systems to install software is inadvisable, though, because their databases are separate. If you install a library using a Debian package and then try to install an RPM that relies on that library, RPM won't realize that the library is already installed and will return an error.

Using RPM

The most popular package manager in the Linux world is RPM. In fact, RPM is available on non-Linux platforms, although it sees less use outside the Linux world. The RPM system provides all the basic tools described in the preceding section, "Package Concepts," such as a package database that allows for checking conflicts and ownership of particular files.

RPM Distributions and Conventions

RPM was developed by Red Hat for its own distribution. Red Hat released the software under the General Public License (GPL), however, so others have been free to use it in their own distributions, and in fact, this is precisely what has happened. Some distributions, such as Mandriva (formerly Mandrake), LinuxPPC, and Yellow Dog, are based on Red Hat, and so they use RPMs as well as many other parts of the Red Hat distribution. Others, such as SuSE and Conectiva, borrow less from the Red Hat template, but they do use RPMs. Of course, all Linux distributions share many common components, so even those that weren't originally based on Red Hat are very similar to it in many ways other than their use of RPM packages. On the other hand, distributions that were originally based on Red Hat have diverged from it over time. As a result, the group of RPM-using distributions shows substantial variability, but all of them are still Linux distributions that provide the same basic tools, such as the Linux kernel, common shells, an X server, and so on.

NOTE   Red Hat has splintered into two distributions: Fedora is the downloadable version favored by home users, students, and businesses on a tight budget. The *Red Hat* name is reserved for the for-pay version of the distribution.

RPM is a cross-platform tool. As noted earlier, some non-Linux Unix systems can use RPM, although most don't use it as their primary package distribution system. RPM supports any CPU architecture. In fact, Red Hat Linux is or has been available for at least five CPUs: x86, AMD64 (aka x86-64), IA-64, Alpha, and SPARC. Among the distributions mentioned earlier, LinuxPPC and Yellow Dog are PowerPC distributions (they run on Apple Power Macs and some non-Apple systems), and SuSE is available on x86, AMD64, PowerPC, and Alpha systems. For the most part, source RPMs are transportable across architectures—you can use the

same source RPM to build packages for *x*86, AMD64, PowerPC, Alpha, SPARC, or any other platform you like. Some programs are actually composed of architecture-independent scripts and so they need no recompilation. There are also documentation and configuration packages that work on any CPU.

The convention for naming RPM packages is as follows:

packagename-a.b.c-x.arch.rpm

Each of the filename components has a specific meaning:

Package name   The first component (*packagename*) is the name of the package, such as samba for the Samba file and print server.

Version number   The second component (*a.b.c*) is the package version number, such as 2.2.7a. The version number doesn't have to be three period-separated numbers, but that's the most common form. The program author assigns the version number.

Build number   The number following the version number (*x*) is the *build number* (also known as the *release number*). This number represents minor changes made by the package maintainer, not by the program author. These changes may represent altered startup scripts or configuration files, changed file locations, added documentation, or patches appended to the original program to fix bugs or to make the program more compatible with the target Linux distribution. Some distribution maintainers add a letter code to the build number to distinguish their packages from those of others. Note that these numbers are *not* comparable across package maintainers—George's build number 5 of a package is *not* necessarily an improvement on Susan's build number 4 of the same package.

Architecture   The final component preceding the .rpm extension (*arch*) is a code for the package's architecture. The i386 architecture code is the most common one; it represents a file compiled for any *x*86 CPU from the 80386 onward. Some packages include optimizations for Pentiums or above (i586 or i686), and non-*x*86 binary packages use codes for their CPUs, such as ppc for PowerPC CPUs or x86_64 for the AMD64 platform. Scripts, documentation, and other CPU-independent packages generally use the noarch architecture code. The main exception to this rule is source RPMs, which use the src architecture code.

As an example of RPM version numbering, the Fedora 3 distribution ships with a Samba package called samba-3.0.10-1.fc3.i386.rpm, indicating that this is build 1.fc3 of Samba 3.0.10, compiled with 386 optimizations. These naming conventions are just that, though—conventions. It's possible to rename a package however you like and it will still install and work. The information in the filename is retained within the package. This fact can be useful if you're ever forced to transfer RPMs using a medium that doesn't allow for long filenames. In fact, early versions of SuSE eschewed long filenames, preferring short filenames such as samba.rpm.

In an ideal world, any RPM package will install and run on any RPM-based distribution that uses an appropriate CPU type. Unfortunately, compatibility issues can crop up from time to time, including these:

▪   Distributions may use different versions of the RPM utilities, as described shortly in "Upgrades to RPM." This problem can completely prevent an RPM from one distribution from being used on another.

- An RPM package designed for one distribution may have dependencies that are unmet in another distribution. A package may require a newer version of a library than is present on the distribution you're using, for instance. This problem can usually be overcome by installing or upgrading the depended-on package, but sometimes this causes problems because the upgrade may break other packages. By rebuilding the package you want to install from a source RPM, you can often work around these problems, but sometimes the underlying source code also needs the upgraded libraries.

- An RPM package may be built to depend on a package of a particular name, such as `samba-client` depending on `samba-common`, but if the distribution you're using has named the package differently, the `rpm` utility will object. You can override this objection by using the `--nodeps` switch, but sometimes the package won't work once installed. Rebuilding from a source RPM may or may not fix this problem.

- Even when a dependency appears to be met, different distributions may include slightly different files in their packages. For this reason, a package meant for one distribution may not run correctly when installed on another distribution. Sometimes installing an additional package will fix this problem.

- Some programs include distribution-specific scripts or configuration files. This problem is particularly acute for servers, which may include startup scripts that go in `/etc/rc.d/init.d` or elsewhere. Overcoming this problem usually requires that you remove the offending script after installing the RPM and either start the server in some other way or write a new startup script, perhaps modeled after one that came with some other server for your distribution.

Despite this list of caveats, mixing and matching RPMs from different distributions usually works reasonably well for most programs, particularly if the distributions are closely related or you rebuild from a source RPM. If you have trouble with an RPM, though, you may do well to try to find an equivalent package that was built with your distribution in mind.

Upgrades to RPM

The earliest versions of RPM were quite primitive by today's standards; for instance, they did not support dependencies. Over time, though, improvements have been made. This fact occasionally causes problems when Red Hat releases a new version of RPM. For instance, Red Hat 7.0 introduced version 4 of the RPM utilities, but version 4 RPM files cannot be installed with most earlier versions of RPM. This led to frustration on the part of many people who used RPM-based distributions in late 2000 because they couldn't use Red Hat 7.0 RPMs on their systems. (RPM 4.3.x is current in mid-2005.)

It's usually possible to overcome such problems by installing a newer version of RPM and upgrading the RPM database. Unfortunately, there's a chicken-and-egg problem, because without the new version of RPM, it's impossible to install the updated version of RPM. Red Hat and many other RPM-based distribution providers frequently do make a version of the next-generation version of RPM available for older systems. In the case of the switch to RPM 4.0 with Red Hat 7.0, Red Hat has made this upgrade available in its Red Hat 6.2 updates area, for instance. After

installing such an upgrade, be sure to type **rpm --rebuilddb** to have the system rebuild your RPM database to conform to the new program's expectations. If you fail to do this, you may be unable to install new programs or access information on old ones.

The *rpm* Command Set

The main RPM utility program is known as rpm. Use this program to install or upgrade a package at the shell prompt. The rpm command has the following syntax:

```
rpm [operation][options] [package-files|package-names]
```

Table 2.1 summarizes the most common rpm operations, and Table 2.2 summarizes the most important options. Be aware, however, that rpm is a very complex tool, so this listing is necessarily incomplete. Tables 2.1 and 2.2 do include information on the most common rpm features, however. For information on operations and options more obscure than those listed in Tables 2.1 and 2.2, see the man pages for rpm. Many of rpm's less-used features are devoted to the creation of RPM packages by software developers.

T A B L E   2 . 1   Common *rpm* Operations

Operation	Description
-i	Installs a package; system must *not* contain a package of the same name.
-U	Installs a new package or upgrades an existing one.
-F or --freshen	Upgrades a package only if an earlier version already exists.
-q	Queries a package—finds if a package is installed, what files it contains, and so on.
-V or -y or --verify	Verifies a package—checks that its files are present and unchanged since installation.
-e	Uninstalls a package.
-b	Builds a binary package, given source code and configuration files; moved to the rpmbuild program with RPM version 4.2.
--rebuild	Builds a binary package, given a source RPM file; moved to the rpmbuild program with RPM version 4.2.
--rebuilddb	Rebuilds the RPM database to fix errors.

T A B L E   2 . 2   Common *rpm* Options

Option	Used with Operations	Description
--root *dir*	Any	Modifies the Linux system having a root directory located at *dir*. This option can be used to maintain one Linux installation discrete from another one (say, during OS installation or emergency maintenance).
--force	-i, -U, -F	Forces installation of a package even when it means overwriting existing files or packages.
-h or --hash	-i, -U, -F	Displays a series of hash marks (#) to indicate the progress of the operation.
-v	-i, -U, -F	Used in conjunction with the -h option to produce a uniform number of hash marks for each package.
--nodeps	-i, -U, -F, -e	Specifies that no dependency checks be performed. Installs or removes the package even if it relies on a package or file that's not present or is required by a package that's not being uninstalled.
--test	-i, -U, -F	Checks for dependencies, conflicts, and other problems without actually installing the package.
--prefix *path*	-i, -U, -F	Sets the installation directory to *path* (works only for some packages).
-a or --all	-q, -V	Queries or verifies all packages.
-f *file* or --file *file*	-q, -V	Queries or verifies the package that owns *file*.
-p *package-file*	-q	Queries the uninstalled RPM *package-file*.
-i	-q	Displays package information, including the package maintainer, a short description, and so on.
-R or --requires	-q	Displays the packages and files on which this one depends.
-l or --list	-q	Displays the files contained in the package.

To use `rpm`, you combine one operation with one or more options. In most cases, you include one or more package names or package filenames as well. (A package filename is a complete filename, but a package name is a shortened version. For instance, a package filename might be `samba-3.0.10-1.fc3.i386.rpm`, while the matching package name is `samba`.) You can issue the `rpm` command once for each package, or you can list multiple packages, separated by spaces, on the command line. The latter is often preferable when you're installing or removing several packages, some of which depend on others in the group. Issuing separate commands in this situation requires that you install the depended-on package first or remove it last, whereas issuing a single command allows you to list the packages on the command line in any order.

Some operations require that you give a package filename, and others require a package name. In particular, `-i`, `-U`, `-F`, and the rebuild operations require package filenames; `-q`, `-V`, and `-e` normally take a package name, although the `-p` option can modify a query (`-q`) operation to work on a package filename.

When you're installing or upgrading a package, the `-U` operation is generally the most useful because it allows you to install the package without manually uninstalling the old one. This one-step operation is particularly helpful when packages contain many dependencies because `rpm` detects these and can perform the operation should the new package fulfill the dependencies provided by the old one.

To use `rpm` to install or upgrade a package, issue a command similar to the following:

```
# rpm -Uvh samba-3.0.10-1.fc3.i386.rpm
```

You could also use `rpm -ivh` in place of `rpm -Uvh` if you don't already have a `samba` package installed.

> **WARNING** It's possible to distribute the same program under different names. In this situation, upgrading may fail, or it may produce a duplicate installation, which can yield bizarre program-specific malfunctions. Red Hat has described a formal system for package naming to avoid such problems, but they still occur occasionally. Therefore, it's best to upgrade a package using a subsequent release provided by the same individual or organization that provided the original.

Verify that the package is installed with the `rpm -qi` command, which displays information such as when and on what computer the binary package was built. Listing 2.1 demonstrates this command. (`rpm -qi` also displays an extended plain-English summary of what the package is, which has been omitted from Listing 2.1.)

Listing 2.1: RPM Query Output

```
$ rpm -qi samba
Name        : samba                       Relocations: (not relocatable)
Version     : 3.0.10                            Vendor: Red Hat, Inc.
Release     : 1.fc3                         Build Date: Fri 17 Dec 2004
➥10:23:20 PM EST
Install Date: Thu 20 Jan 2005 03:43:12 PM EST     Build Host:
```

➥tweety.build.redhat.com
Group   : System Environment/Daemons   Source RPM:
 ➥samba-3.0.10-1.fc3.src.rpm
Size   : 24865876   License: GNU GPL Version 2
Signature   : DSA/SHA1, Mon 20 Dec 2004 01:42:00 PM EST, Key
 ➥ID b44269d04f2a6fd2
Packager   : Red Hat, Inc. <http://bugzilla.redhat.com/bugzilla>
URL   : http://www.samba.org/
Summary   : The Samba SMB server.

EXERCISE 2.1

Managing Packages Using RPM

This exercise runs you through the process of managing packages using the rpm utility. To manage packages, follow these steps:

1. Log into the Linux system as a normal user.

2. Acquire a package to use for testing purposes. You can try using a package from your distribution that you know you haven't installed, but if you try a random package, you may find it's already installed or you may find it has unmet dependencies. This lab uses as an example the installation of zsh-4.2.0-3.i386.rpm, a shell that's not installed by default on most systems, from the third Fedora 3 CD onto a Fedora 3 system. You must adjust the commands as necessary if you use another RPM file in your tests.

3. Launch an xterm from the desktop environment's menu system if you used a GUI login.

4. Acquire root privileges. You can do this by typing **su** in an xterm, by selecting Session ➢ New Root Console from a Konsole window, or by using sudo (if it's configured) to run the commands in the following steps.

5. Type **rpm -q zsh** to verify that the package is not currently installed. The system should respond with the message package zsh is not installed.

6. Type **rpm -qpi zsh-4.0.7-1.1.i386.rpm**. (You'll need to add a complete path to the package file if it's not in your current directory.) The system should respond by displaying information about the package, such as the version number, the vendor, the hostname of the machine on which it was built, and a package description.

7. Type **rpm -ivh zsh-4.0.7-1.1.i386.rpm**. The system should install the package and display a series of hash marks (#) as it does so.

8. Type **rpm -q zsh**. The system should respond with the complete package name, including the version and build numbers. This response verifies that the package is installed.

9. Type **zsh**. This launches a Z shell, which functions much like the more common bash and tcsh shells. You're likely to see your command prompt change slightly, but you can issue most of the same commands you can use with bash or tcsh.

10. Type **rpm -V zsh**. The system should not produce any output, just display a new command prompt. The verify (-V or --verify) command checks the package files against data stored in the database. Immediately after installation, most packages should show no deviations. (A handful of packages will be modified during installation, but zsh isn't one of them.)

11. Type **rpm -e zsh**. The system shouldn't produce any output—just a new command prompt. This command removes the package from the system, though. Note that you're removing the zsh package while running the zsh program. Linux continues to run the zsh program you're using, but you'll be unable to launch new instances of the program. Some programs may misbehave if you do this because files will be missing after you remove the package.

12. Type **exit** to exit from zsh and return to your normal shell.

13. Type **rpm -q zsh**. The system should respond with a package zsh is not installed error because you've just uninstalled it.

Extracting Data from RPMs

Occasionally you may want to extract data from RPMs without actually installing the package. For instance, this can be a good way to retrieve the original source code from a source RPM for compiling the software without the help of the RPM tools or to retrieve fonts or other non-program data for use on a non-RPM system.

RPM files are actually modified cpio archives. Thus, converting the files into cpio files is relatively straightforward, whereupon you can use cpio to retrieve the individual files. To do this job, you need to use the rpm2cpio program, which ships with most Linux distributions. (You can use this tool even on distributions that don't use RPM.) This program takes a single argument—the name of the RPM file—and outputs the cpio archive on standard output. Thus, if you want to create a cpio archive file, you must redirect the output:

```
$ rpm2cpio samba-3.0.10-1.fc3.src.rpm > samba-3.0.10-1.fc3.cpio
```

> **NOTE** The redirection operator, >, is described in more detail in Chapter 1, "Linux Command-Line Tools," as is the pipe operator (|), which is mentioned shortly.

You can then extract the data using cpio, which takes the -i option to extract an archive and --make-directories to create directories:

```
$ cpio -i --make-directories < samba-3.0.10-1.fc3.cpio
```

Alternatively, you can use a pipe to link these two commands together without creating an intermediary file:

```
$ rpm2cpio samba-3.0.10-1.fc3.src.rpm | cpio -i --make-directories
```

In either case, the result is an extraction of the files in the archive in the current directory. In the case of binary packages, this is likely to be a series of subdirectories that mimic the layout of the Linux root directory—that is, `usr`, `lib`, `etc`, and so on, although precisely which directories are included depends on the package. For a source package, the result of the extraction process is likely to be a source code tarball, a `.spec` file (which holds information RPM uses to build the package), and perhaps some patch files.

> **TIP** When extracting data from an RPM file using `rpm2cpio` and `cpio`, create a holding subdirectory and then extract the data into this subdirectory. This practice will ensure that you can find all the files. If you extract files in your home directory, some of them might get lost. If you extract files as `root` in the root (/) directory, they could conceivably overwrite files that you want to keep.

Another option for extracting data from RPMs is to use `alien`, which is described later, in "Converting between Package Formats." This program can convert an RPM into a Debian package or a tarball.

Real World Scenario

Using Higher-Level RPM Utilities

To ease system administration, many distributions now provide higher-level RPM tools. The Yum program (`http://linux.duke.edu/projects/yum/`) is popular in this respect and is the basis for GUI tools such as the Red Hat and Fedora Update Agent. Another option is to use the Advanced Packaging Tool (APT) from Debian; a port for RPM versions is known as `apt4rpm` and is available from `http://apt4rpm.sourceforge.net`.

The upcoming section, "Using apt-get," describes using APT, and that description is applicable to `apt4rpm`, although you'll need to install APT using more conventional means and locate an appropriate repository site. (The `apt4rpm` home page can help with those tasks.)

One advantage of higher-level utilities is that they permit easy installation of new software without juggling your CD-ROMs; just type a command, such as **apt-get install someprog**, and the `someprog` package will be installed on your system (if it exists). Another advantage is that these tools help automate system updates. For instance, typing **apt-get update; apt-get dist-upgrade** will upgrade all your packages to the latest versions. Such mass upgrades are not without their risks (a buggy program could cause problems), but on the whole, easy system updates are a boon for system administration.

RPM Configuration Files

Ordinarily, you needn't explicitly configure RPM; distributions that use RPM configure it in reasonable ways by default. Sometimes, though, you might want to tweak a few details, particularly if you routinely build source RPM packages and want to optimize the output for your system. To do so, you typically edit an RPM configuration file.

The main RPM configuration file is `/usr/lib/rpm/rpmrc`. This file sets a variety of options, mostly related to the CPU optimizations used when compiling source packages. This file, though, should not be edited; instead, you should create and edit `/etc/rpmrc` (to make global changes) or `~/.rpmrc` (to make changes on a per-user basis). The main reason to create such a file is to implement architecture optimizations—for instance, to optimize your code for your CPU model by passing appropriate compiler options when you build a source RPM into a binary RPM. This is done with the `optflags` line:

```
optflags: athlon -O2 -g -march=i686
```

This line tells RPM to pass the `-O2 -g -march-i686` options to the compiler whenever building for the `athlon` platform. Although RPM can determine your system's architecture, the `optflags` line by itself isn't likely to be enough to set the correct flags. Most default `rpmrc` files include a series of `buildarchtranslate` lines that cause `rpmbuild` (or `rpm` for older versions of RPM) to use one set of optimizations for a whole family of CPUs. For *x86* systems, these lines typically look like this:

```
buildarchtranslate: athlon: i386
buildarchtranslate: i686: i386
buildarchtranslate: i586: i386
buildarchtranslate: i486: i386
buildarchtranslate: i386: i386
```

These lines tell RPM to translate the `athlon`, `i686`, `i586`, `i486`, and `i386` CPU codes to use the `i386` optimizations. This effectively defeats the purpose of any CPU-specific optimizations you create on the `optflags` line for your architecture, but it guarantees that the RPMs you build will be maximally portable. To change matters, you must alter the line for your CPU type, as returned when you type **uname -p**. For instance, on an Athlon-based system, you might enter the following line:

```
buildarchtranslate: athlon: athlon
```

Thereafter, when you rebuild a source RPM, the system uses the appropriate Athlon optimizations. The result can be a slight performance boost on your own system, but reduced portability—depending on the precise optimizations you choose, such packages might not run on non-Athlon CPUs. (Indeed, you may not even be able to install them on non-Athlon CPUs!)

RPM Compared to Other Package Formats

RPM is a very flexible package management system. In most respects, it's comparable to Debian's package manager, and it offers many more features than tarballs do. When compared to Debian packages, the greatest strength of RPMs is probably their ubiquity. Many software packages are available in RPM form from their developers and/or from distribution maintainers.

> **NOTE** Distribution packagers frequently modify the original programs in order to make them integrate more smoothly into the distribution as a whole. For instance, distribution-specific startup scripts may be added, program binaries may be relocated from default /usr/local subdirectories, and program source code may be patched to fix bugs or add features. Although these changes can be useful, you may not want them, particularly if you're using a program on a distribution other than the one for which the package was intended.

The fact that there are so many RPM-based distributions can be a boon. You may be able to use an RPM intended for one distribution on another, although as noted earlier, this isn't certain. In fact, this advantage can turn into a drawback if you try to mix and match too much—you can wind up with a mishmash of conflicting packages that can be very difficult to disentangle.

> **TIP** The RPMFind website, http://rpmfind.net, is an extremely useful resource when you want to find an RPM of a specific program. Another site with similar characteristics is Fresh RPMs, http://www.freshrpms.net. These sites include links to RPMs built by programs' authors, specific distributions' RPMs, and those built by third parties.

Compared to tarballs, RPMs offer much more sophisticated package management tools. This can be important when you're upgrading or removing packages and also for verifying the integrity of installed packages. On the other hand, although RPMs are very common in the Linux world, they're less common on other platforms. Therefore, you're more likely to find tarballs of generic Unix source code, and tarballs are preferred if you've written a program that you intend to distribute for other platforms.

Using Debian Packages

In their overall features, Debian packages are similar to RPMs, but the details of operation for each differ, and Debian packages are used on different distributions than are RPMs. Because each system uses its own database format, RPMs and Debian packages aren't interchangeable without converting formats. Using Debian packages requires knowing how to use the dpkg, dselect, and apt-get commands. A few other commands can also be helpful.

Debian Distributions and Conventions

As the name implies, Debian packages originated with the Debian distribution. Since that time, the format has been adopted by several other distributions, including Libranet, Ubuntu, and Xandros. Such distributions are derived from the original Debian, which means that packages from the original Debian are likely to work well on other Debian-based systems. Although Debian doesn't emphasize flashy GUI installation or configuration tools, its derivatives—particularly Xandros—add GUI configuration tools to the base Debian system, which makes these distributions more appealing to Linux novices. The original Debian favors a system that's as bug free as possible, and it tries to adhere strictly to open-source software principles rather than invest effort in GUI configuration tools. The original Debian is unusual in that it's maintained by volunteers who are motivated by the desire to build a product they want to use rather than by a company that is motivated by profits.

Like RPM, the Debian package format is neutral with respect to both OS and CPU type. Debian packages are extremely rare outside Linux, although efforts are under way to create a Debian distribution that uses the GNU Hurd kernel rather than the Linux kernel. Such a distribution would not be Linux but would closely resemble Debian GNU/Linux in operation and configuration.

The original Debian distribution has been ported to many different CPUs, including $x86$, IA-64, PowerPC, Alpha, $680x0$, MIPS, and SPARC. The original architecture was $x86$, and subsequent ports exist at varying levels of maturity. Derivative distributions generally work only on $x86$ systems, but this could change in the future.

Debian packages follow a naming convention similar to those for RPMs, but Debian packages sometimes omit codes in the filename to specify a package's architecture, particularly on $x86$ packages. When these codes are present, they may differ from RPM conventions. For instance, a filename ending in `i386.deb` indicates an $x86$ binary, `powerpc.deb` is a PowerPC binary, and `all.deb` indicates a CPU-independent package, such as documentation or scripts. As with RPM files, this file-naming convention is only that—a convention. You can rename a file as you see fit, to either include or omit the processor code. There is no code for Debian source packages because, as described in the upcoming section, "Debian Packages Compared to Other Package Formats," Debian source packages actually consist of several separate files.

The *dpkg* Command Set

Debian packages are incompatible with RPM packages, but the basic principles of operation are the same across both package types. Like RPMs, Debian packages include dependency information, and the Debian package utilities maintain a database of installed packages, files, and so on. You use the `dpkg` command to install a Debian package. This command's syntax is similar to that of `rpm`:

```
dpkg [options][action] [package-files|package-name]
```

The `action` is the action to be taken; common actions are summarized in Table 2.3. The options (Table 2.4) modify the behavior of the action, much like the options to `rpm`.

TABLE 2.3   *dpkg* Primary Actions

Action	Description
-i or --install	Installs a package
--configure	Reconfigures an installed package: runs the post-installation script to set site-specific options
-r or --remove	Removes a package, but leaves configuration files intact
-P or --purge	Removes a package, including configuration files
-p or --print-avail	Displays information about an installed package
-I or --info	Displays information about an uninstalled package file
-l *pattern* or --list *pattern*	Lists all installed packages whose names match *pattern*
-L or --listfiles	Lists the installed files associated with a package
-S *pattern* or --search *pattern*	Locates the package(s) that own the file(s) specified by *pattern*
-C or --audit	Searches for partially installed packages and suggests what to do with them

TABLE 2.4   Options for Fine-Tuning *dpkg* Actions

Option	Used with Actions	Description
--root=*dir*	All	Modifies the Linux system using a root directory located at *dir*. Can be used to maintain one Linux installation discrete from another one, say during OS installation or emergency maintenance.
-B or --auto-deconfigure	-r	Disables packages that rely on one that is being removed.
--force-*things*	Assorted	Forces specific actions to be taken. Consult the dpkg man page for details of *things* this option does.
--ignore-depends=*package*	-i, -r	Ignores dependency information for the specified package.

TABLE 2.4   Options for Fine-Tuning *dpkg* Actions *(continued)*

Option	Used with Actions	Description
`--no-act`	`-i, -r`	Checks for dependencies, conflicts, and other problems without actually installing or removing the package.
`--recursive`	`-i`	Installs all packages that match the package name wildcard in the specified directory and all sub-directories.
`-G`	`-i`	Doesn't install the package if a newer version of the same package is already installed.
`-E` or `--skip-same-version`	`-i`	Doesn't install the package if the same version of the package is already installed.

As with `rpm`, `dpkg` expects a package name in some cases and a package filename in others. Specifically, `--install` (`-i`) and `--info` (`-I`) both require the package filename, but the other commands take the shorter package name.

As an example, consider the following command, which installs the `samba-common_3.0.10-1_powerpc.deb` package:

```
# dpkg -i samba-common_3.0.10-1_powerpc.deb
```

If you're upgrading a package, you may need to remove an old package before installing the new one. To do this, use the `-r` option to `dpkg`, as in the following:

```
# dpkg -r samba
```

To find information on an installed package, use the `-p` parameter to `dpkg`, as shown in Listing 2.2. This listing omits an extended English description of what the package does.

Listing 2.2: *dpkg* Package Information Query Output

```
$ dpkg -p samba-common
Package: samba-common
Priority: optional
Section: net
Installed-Size: 4824
Maintainer: Eloy A. Paris <peloy@debian.org>
Architecture: powerpc
Source: samba
Version: 3.0.10-1
Replaces: samba (<< 2.999+3.0.alpha21-4)
```

```
Depends: debconf, libpam-modules, libc6 (>= 2.3.2.ds1-4), libcomerr2➡
 (>= 1.33-3), libkrb53 (>= 1.3.2), libldap2 (>= 2.1.17-1)
Filename: pool/updates/main/s/samba/samba-common_2.2.3a-12_powerpc.deb
Size: 2103052
MD5sum: e4b852940d6bdce313cb3e7b668e2c21
```

Debian-based systems often use a pair of somewhat higher-level utilities, apt-get and dselect, to handle package installation and removal. These utilities are described in the next couple of sections. Their interfaces can be very useful when you want to install several packages, but dpkg is often more convenient when manipulating just one or two packages. Because dpkg can take package filenames as input, it's also the preferred method of installing a package that you download from an unusual source or create yourself.

Using *apt-get*

Another option for Debian package management is the Advanced Packaging Tool (APT) utilities, and particularly apt-get. This tool enables you to perform easy upgrades of packages, especially if you have a fast Internet connection. Debian-based systems include a file, /etc/apt/sources.list, that specifies locations from which important packages can be obtained. If you installed the OS from a CD-ROM drive, this file will initially list directories on the installation CD-ROM in which packages can be found. There are also likely to be a few lines near the top, commented out with hash marks (#), indicating directories on an FTP or website from which you can obtain updated packages. (These lines may be uncommented if you did a network install initially.)

> Although APT is most strongly associated with Debian systems, a port to RPM-based systems is also available. Check http://apt4rpm.sourceforge.net for information on this port.

> **WARNING** Don't add a site to /etc/apt/sources.list unless you're sure it can be trusted. The apt-get utility does automatic and semiautomatic upgrades, so if you add a network source to sources.list and that source contains unreliable programs or programs with security holes, your system will become vulnerable after upgrading via apt-get.

The apt-get utility works by obtaining information on available packages from the sources listed in /etc/apt/sources.list and then using that information to upgrade or install packages. The syntax is similar to that of dpkg:

```
apt-get [options][command] [package-names]
```

Table 2.5 lists the `apt-get` commands, and Table 2.6 lists the most commonly used options. In most cases, you won't actually use *any* options with `apt-get`, just a single command and possibly one or more package names. One particularly common use of this utility is to keep your system up-to-date with any new packages. The following two commands will accomplish this goal if `/etc/apt/sources.list` includes pointers to up-to-date file archive sites:

```
# apt-get update
# apt-get dist-upgrade
```

T A B L E   2 . 5   *apt-get* Commands

Command	Description
update	Obtains updated information on packages available from the installation sources listed in `/etc/apt/sources.list`.
upgrade	Upgrades all installed packages to the newest versions available, based on locally stored information on available packages.
dselect-upgrade	Performs any changes in package status (installation, removal, etc.) left undone after running `dselect`.
dist-upgrade	Similar to upgrade, but performs "smart" conflict resolution to avoid upgrading a package if that would break a dependency.
install	Installs a package by package name (not by package filename), obtaining the package from the source that contains the most up-to-date version.
remove	Removes a specified package by package name.
source	Retrieves the newest available source package file by package filename using information on available packages and installation archives listed in `/etc/apt/sources.list`.
check	Checks the package database for consistency and broken package installations.
clean	Performs housekeeping to help clear out information on retrieved files from the Debian package database. If you don't use `dselect` for package management, run this from time to time in order to save disk space.
autoclean	Similar to `clean`, but removes information only on packages that can no longer be downloaded.

T A B L E   2 . 6   Most Useful *apt-get* Options

Option	Used with Commands	Description
-d or --download-only	upgrade, dselect-upgrade, install, source	Downloads package files but does not install them.
-f or --fix-broken	install, remove	Attempts to fix a system on which dependencies are unsatisfied.
-m, --ignore-missing, or --fix-missing	upgrade, dselect-upgrade, install, remove, source	Ignores all package files that can't be retrieved (because of network errors, missing files, or the like).
-q or --quiet	All	Omits some progress indicator information. May be doubled (for instance, -qq) to produce still less progress information.
-s, --simulate, --just-print, --dry-run, --recon, or --no-act	All	Performs a simulation of the action without actually modifying, installing, or removing files.
-y, --yes, or --assume-yes	All	Produces a "yes" response to any yes/no prompt in installation scripts.
-b, --compile, or --build	source	Compiles a source package after retrieving it.
--no-upgrade	install	Causes apt-get to *not* upgrade a package if an older version is already installed.

WARNING   If you use apt-get to automatically upgrade all packages on your system, you are effectively giving control of your system to the distribution maintainer. Although Debian or other distribution maintainers are unlikely to try to break into your computer in this way, an automatic update with minimal supervision on your part could easily break something on your system, particularly if you've obtained packages from unusual sources in the past.

EXERCISE 2.2

Managing Debian Packages

In this exercise, you'll familiarize yourself with the Debian package system. To manage packages, follow these steps:

1. Log into the Linux system as a normal user.

2. Acquire a package to use for testing purposes. You can try using a package from your distribution that you know you haven't installed, but if you try a random package, you may find it's already installed or you may find it has unmet dependencies. This lab uses as an example the installation of zsh_4.0.4-33_powerpc.deb, a shell that's not installed by default on most systems, from the first CD-ROM of a Debian 3.0 for a PowerPC system. You must adjust the commands as necessary if you use another package in your tests. If you're installing on an *x*86 system, the package filename will differ as well.

3. Launch an xterm from the desktop environment's menu system if you used a GUI login.

4. Acquire root privileges. You can do this by typing **su** in an xterm, by selecting Session ➢ New Root Console from a Konsole window, or by using sudo (if it's configured) to run the commands in the following steps.

5. Type **dpkg -L zsh** to verify that the package is not currently installed. This command responds with a list of files associated with the package if it's installed or with an error that reads Package `zsh' is not installed if it's not.

6. Type **dpkg -I zsh_4.0.4-33_powerpc.deb**. (You'll need to add a complete path to the package file if it's not in your current directory.) The system should respond by displaying information about the package, such as the version number, dependencies, the name of the package maintainer, and a package description.

7. Type **dpkg -i zsh_4.0.4-33_powerpc.deb**. The system should install the package and display a series of lines summarizing its actions as it does so.

8. Type **dpkg -p zsh**. The system should respond with information on the package similar to that displayed in step 6.

9. Type **zsh**. This launches a Z shell, which functions much like the more common bash and tcsh shells. You're likely to see your command prompt change slightly, but you can issue most of the same commands you can use with bash or tcsh.

10. Type **dpkg -P zsh**. This command removes the package from the system, including configuration files. It will produce a series of warnings about nonempty directories that it couldn't remove. Note that you're removing the zsh package while running the zsh program. Linux continues to run the zsh program you're using, but you'll be unable to launch new instances of the program. Some programs may misbehave because files will be missing after you remove the package.

11. Type **exit** to exit from zsh and return to your normal shell.

12. Type **dpkg -L zsh**. The system should respond with a Package `zsh' is not installed error because you've just uninstalled it.

13. Type **apt-get install zsh** to install zsh using the Advanced Package Tools (APT) system. Depending on your configuration, the system may download the package from an Internet site or ask you to insert a CD-ROM. If it asks for a CD-ROM, insert it and press the Enter key. The system should install the package.

14. Type **dpkg -p zsh**. The system should respond with information on the package similar to that displayed in step 6 or 8.

15. Type **dpkg -P zsh**. This command removes the package from the system, as described in step 10.

Using *dselect*

The dselect program is a high-level package browser. Using it, you can select packages to install on your system from the APT archives defined in /etc/apt/sources.list, review the packages that are already installed on your system, uninstall packages, and upgrade packages. Overall, dselect is a very powerful tool, but it can also be an intimidating one to the uninitiated because it presents a lot of options that aren't obvious, using a text-mode interactive user interface.

Although dselect supports a few command-line options, they're mostly obscure or minor (such as options to set the color scheme). Consult dselect's man page for details. To use the program, type **dselect**. The result is the dselect main menu, as shown running in a KDE Konsole window in Figure 2.1.

The main dselect menu presents six options that enable you to perform various actions:

Set the access method   The first option, [A]ccess, lets you tell dselect what sources to use for software. The best option is often to use APT, but if you don't want to configure /etc/sources.list, you can point dselect directly at your installation CD-ROM, a Network File System (NFS) server, or some other source.

Update software lists   You can tell dselect to update its list of available packages with its [U]pdate option. Doing so before you install new software from a network source is generally advisable so that you install the latest software available.

Select software   The [S]elect option is the one that's most intimidating. When you pick this option, dselect displays an instruction screen and then shows package lists. Select packages to install or uninstall by pressing the + and - keys on your keyboard, respectively. You can search for packages by pressing the slash (/) key followed by a search term. When you're done, press the Enter key to initiate the changes or press the X key to abort the operation.

Install software   The [I]nstall option adds any software to your system that's been marked for installation but not yet installed.

Configure software   Debian packages sometimes include scripts to configure software after the package has been installed. Selecting the [C]onfig option runs these scripts. Some of these scripts require interaction to enable you to pick sensible defaults for your system, so be prepared to answer some questions.

Remove software   If you opted to remove software, the [R]emove option will do so.

Exit   The [Q]uit option exits from the program.

Overall, dselect is a useful tool, particularly if you need to locate software but don't know its exact name—the ability to browse and search the available packages can be a great boon. Unfortunately, the huge package list can be intimidating, particularly if you're not familiar with the dselect package browser interface.

> A somewhat friendlier dselect-like tool is Synaptic. This program has functionality that's similar to dselect, but Synaptic is an X-based GUI tool with a point-and-click interface. This design makes for easier package selection if you're not familiar with the dselect package browser.

FIGURE 2.1   The dselect utility provides access to APT features using a menu system.

Reconfiguring Packages

Debian packages often provide more extensive initial setup options than do their RPM counterparts. Frequently, the install script included in the package asks a handful of questions, such as asking for the name of an outgoing mail relay system for a mail server program. These questions help the system set up a standardized configuration that's nonetheless been customized for your system.

In the course of your system administration, you may alter the configuration files for a package. If you do this and find you've made a mess of things, you may want to revert to the initial standard configuration. To do so, you can use the `dpkg-reconfigure` program, which runs the initial configuration script for the package you specify:

```
# dpkg-reconfigure samba
```

This command reconfigures the `samba` package, asking the package's initial installation questions and restarting the Samba daemons. Once this is done, the package should be in something closer to its initial state.

Debian Packages Compared to Other Package Formats

The overall functionality of Debian packages is similar to that of RPMs, although there are differences. Debian source packages are not single files; they're groups of files—the original source tarball, a patch file that's used to modify the source code (including a file that controls the building of a Debian package), and a `.dsc` file that contains a digital "signature" to help verify the authenticity of the collection. The Debian package tools can combine these and compile the package to create a Debian binary package. This structure makes Debian source packages slightly less convenient to transport because you must move at least two files (the tarball and patch file; the `.dsc` file is optional) rather than just one. Debian source packages also support just one patch file, whereas RPM source packages may contain multiple patch files. Although you can certainly combine multiple patch files into one, doing so makes it less clear where a patch comes from, thus making it harder to back out of any given change.

These source package differences are mostly of interest to software developers, however. As a system administrator or end user, you need not normally be concerned with them unless you must recompile a package from a source form—and even then, the differences between the formats need not be overwhelming. The exact commands and features used by each system differ, but they accomplish similar overall goals.

Because all distributions that use Debian packages are derived from Debian, they tend to be more compatible with one another (in terms of their packages) than RPM-based distributions are. In particular, Debian has defined details of its system startup scripts and many other features to help Debian packages install and run on any Debian-based system. This helps Debian-based systems avoid the sorts of incompatibilities in startup scripts that can cause problems using one distribution's RPMs on another distribution. Of course, some future distribution could violate Debian's guidelines for these matters, so this advantage isn't guaranteed to hold over time.

As a practical matter, it can be harder to locate Debian packages than RPM packages for some more exotic programs. Nonetheless, Debian maintains a good collection at `http://`

`www.debian.org/distrib/packages`, and some program authors make Debian packages available as well. If you can find an RPM but not a Debian package, you may be able to convert the RPM to Debian format using a program called `alien`, as described shortly in "Converting between Package Formats." If all else fails, you can use a tarball, but you'll lose the advantages of the Debian package database.

Configuring Debian Package Tools

With the exception of the APT sources list mentioned earlier, Debian package tools don't usually require configuration. Debian (and its derivative distributions) install reasonable defaults. On rare occasion, though, you might want to adjust some of these defaults. Doing so requires that you know where to look for them.

The main configuration file for `dpkg` is `/etc/dpkg/dpkg.cfg` or `~/.dpkg.cfg`. This file contains `dpkg` options, as summarized in Table 2.4, but without the leading dashes. For instance, to have `dpkg` always perform a test run rather than actually install a package, you'd create a `dpkg.cfg` file that contains one line:

```
no-act
```

For APT, the main configuration file you're likely to modify is `/etc/apt/sources.list`, which is described earlier, in "Using `apt-get`." Beyond this file is `/etc/apt/apt.conf`, which controls APT and `dselect` options. As with `dpkg.cfg`, chances are you won't need to modify `apt.conf`. If you do need to make changes, the format is more complex and is modeled after those of the Internet Software Consortium's (ISC's) Dynamic Host Configuration Protocol (DHCP) and Berkeley Internet Name Domain (BIND) servers' configuration files. Specifically, options are grouped together by open and close curly braces (`{}`):

```
APT
{
  Get
  {
    Download-Only "true";
  };
};
```

These lines are equivalent to permanently setting the `--download-only` option described in Table 2.6. You can, of course, set many more options. For details, consult `apt.conf`'s `man` page. You may also want to review the sample configuration file, `/usr/share/doc/apt/examples/apt.conf`. (The working `/etc/apt/apt.conf` file is typically extremely simple and therefore not very helpful as an example.)

You should be aware that Debian's package tools rely on various files in the `/var/lib/dpkg` directory tree. These files maintain lists of available packages, lists of installed packages, and so on. In other words, this directory tree is effectively the Debian installed file database. As such, you should be sure to back up this directory when you perform system backups and be careful about modifying its contents.

Converting between Package Formats

Sometimes you're presented with a package file in one format but you want to use another format. This is particularly common when you use a Debian-based distribution and can only find tarballs or RPM files of a package. When this happens, you can keep looking for a package file in the appropriate format, install the tools for the foreign format, create a package from a source tarball using the standard RPM or Debian tools, or convert between package formats with a utility like `alien`.

This section focuses on this last option. The `alien` program comes with Debian and a few other distributions but may not be installed by default. If it's not installed on your system, install it by typing **apt-get install alien** on a system that uses APT, or use the RPM Find or Debian package websites to locate it. This program can convert between RPM packages, Debian packages, Stampede packages (used by Stampede Linux), and tarballs. There are some caveats, however. For one thing, `alien` requires that you have appropriate package manager software installed—for instance, both RPM and Debian to convert between these formats. The `alien` utility doesn't always convert all dependency information completely correctly. When converting from a tarball, `alien` copies the files directly as they had been in the tarball, so `alien` works only if the original tarball has files that should be installed off the root (/) directory of the system.

> **NOTE** Although `alien` requires both RPM and Debian package systems to be installed to convert between these formats, `alien` doesn't use the database features of these packages unless you use the `--install` option. The presence of a foreign package manager isn't a problem as long as you don't use it to actually install software that might duplicate or conflict with software installed with your primary package manager.

The basic syntax of `alien` is as follows:

```
alien [options] file[...]
```

The most important options are `--to-deb`, `--to-rpm`, `--to-slp`, and `--to-tgz`, which convert to Debian, RPM, Stampede, or tarball format, respectively. (If you omit the destination format, `alien` assumes you want a Debian package.) The `--install` option installs the converted package and removes the converted file. Consult the `alien man` page for additional options.

For instance, suppose you have a Debian package called `someprogram-1.2.3-4_i386.deb` but you want to create an RPM from this. You could issue the following command to create an RPM called `someprogram-1.2.3-5.i386.rpm`:

```
# alien --to-rpm someprogram-1.2.3-4_i386.deb
```

If you use a Debian-based system and want to install a tarball but keep a record of the files it contains in your Debian package database, you can do so with the following command:

```
# alien --install binary-tarball.tar.gz
```

It's important to remember that converting a tarball converts the files in the directory structure of the original tarball using the system's root directory as the base. Therefore, you may need to unpack the tarball, juggle files around, and repack it to get the desired results *prior to* installing the tarball with `alien`. For instance, suppose you've got a binary tarball that creates a directory called `program-files`, with `bin`, `man`, and `lib` directories under this. The intent might have been to unpack the tarball in `/usr` or `/usr/local` and create links for critical files. To convert this tarball to an RPM, you might issue the following commands:

```
# tar xvfz program.tar.gz
# mv program-files usr
# tar cvfz program.tgz usr
# rm -r usr
# alien --to-rpm program.tgz
```

By renaming the `program-files` directory to `usr` and creating a new tarball, you've created a tarball that, when converted to RPM format, will have files in the locations you want—`/usr/bin`, `/usr/man`, and `/usr/lib`. You might need to perform more extensive modifications, depending on the contents of the original tarball.

Package Dependencies and Conflicts

Although package installation often proceeds smoothly, there are times when it doesn't. The usual sources of problems relate to unsatisfied dependencies or conflicts between packages. The RPM and Debian package management systems are intended to help you locate and resolve such problems, but on occasion (particularly when mixing packages from different vendors), they can actually cause problems. In either event, it pays to recognize these errors and know how to resolve them.

Real and Imagined Package Dependency Problems

Package dependencies and conflicts can arise for a variety of reasons, including the following:

Missing libraries or support programs   One of the most common dependency problems is caused by a missing support package. For instance, all K Desktop Environment (KDE) programs rely on Qt, a widget set that provides assorted GUI tools. If Qt isn't installed, you won't be able to install any KDE packages using RPMs or Debian packages. Libraries—support code that can be used by many different programs as if it were part of the program itself—are particularly common sources of problems in this respect.

Incompatible libraries or support programs   Even if a library or support program is installed on your system, it may be the wrong version. For instance, if a program requires Qt 3.2, the presence of Qt 2.2 won't do much good. Fortunately, Linux library naming conventions enable you to install multiple versions of a library in case you have programs with competing requirements.

Duplicate files or features   Conflicts arise when one package includes files that are already installed and that belong to another package. Occasionally, broad features can conflict as well, as in two web server packages. Feature conflicts are usually accompanied by name conflicts. Conflicts are most common when mixing packages intended for different distributions because distributions may split files up across packages in different ways.

Mismatched names   RPM and Debian package management systems give names to their packages. These names don't always match across distributions. For this reason, if one package checks for another package by name, the first package may not install on another distribution, even if the appropriate package is installed, because that target package has a different name.

Some of these problems are very real and serious. Missing libraries, for instance, must be installed. (Sometimes, though, a missing library isn't quite as missing as it seems, as described in the upcoming section "Forcing the Installation.") Others, like mismatched package names, are artifacts of the packaging system. Unfortunately, it's not always easy to tell into which category a conflict fits. When using a package management system, you may be able to use the error message returned by the package system, along with your own experience with and knowledge of specific packages, to make a judgment. For instance, if RPM reports that you're missing a slew of libraries with which you're unfamiliar, you'll probably have to track down at least one package—unless you know you've installed the libraries in some other way, in which case you may want to force the installation.

Workarounds to Package Dependency Problems

When you encounter an unmet package dependency or conflict, what can you do about it? There are several approaches to these problems. Some of these approaches work well in some situations but not others, so you should review the possibilities carefully. The options include forcing the installation, modifying your system to meet the dependency, rebuilding the problem package from source code, and finding another version of the problem package.

Forcing the Installation

One approach is to ignore the issue. Although this sounds risky, in some cases involving failed RPM or Debian dependencies, it's appropriate. For instance, if the dependency is on a package that you installed by compiling the source code yourself, you can safely ignore the dependency. When using `rpm`, you can tell the program to ignore failed dependencies by using the `--nodeps` parameter thus:

```
# rpm -i apackage.rpm --nodeps
```

You can force installation over some other errors, such as conflicts with existing packages, by using the `--force` parameter:

```
# rpm -i apackage.rpm --force
```

> **WARNING**   Do *not* use --nodeps or --force as a matter of course. Ignoring the dependency checks can lead you into trouble, so you should use these options only when you need to do so. In the case of conflicts, the error messages you get when you first try to install without --force will tell you which packages' files you'll be replacing, so be sure you back them up or are prepared to reinstall the package in case of trouble.

If you're using dpkg, you can use the --ignore-depend=*package*, --force-depends, and --force-conflicts parameters to overcome dependency and conflict problems in Debian-based systems. Because there's less deviation in package names and requirements among Debian-based systems, though, these options are less often needed on such systems.

Upgrading or Replacing the Depended-On Package

Officially, the proper way to overcome a package dependency problem is to install, upgrade, or replace the depended-upon package. If a program requires, say, Qt 3.3 or greater, you should upgrade an older version (such as 3.2) to 3.3. To perform such an upgrade, you'll need to track down and install the appropriate package. This usually isn't too difficult if the new package you want comes from a Linux distribution; the appropriate depended-on package should come with the same distribution.

One problem with this approach is that packages intended for different distributions sometimes have differing requirements. If you run Distribution A and install a package that was built for Distribution B, the package will express dependencies in terms of Distribution B's files and versions. The appropriate versions may not be available in a form intended for Distribution A, and by installing Distribution B's versions, you can sometimes cause conflicts with other Distribution A packages. Even if you install the upgraded package and it works, you could run into problems in the future when it comes time to install some other program or upgrade the distribution as a whole—the upgrade installer might not recognize Distribution B's package or might not be able to upgrade to its own newer version.

Rebuilding the Problem Package

Some dependencies result from the libraries and other support utilities installed on the computer that compiled the package, not from requirements in the underlying source code. If the software is recompiled on a system that has different packages, the dependencies will change. Therefore, rebuilding a package from source code can overcome at least some dependencies.

If you use an RPM-based system, the command to rebuild a package is straightforward: You call rpm or rpmbuild with the name of the source package and use --rebuild, as follows:

```
# rpmbuild --rebuild packagename-version.src.rpm
```

Of course, to do this you must have the source RPM for the package. This can usually be obtained from the same location as the binary RPM. When you execute this command, rpm extracts the source code and executes whatever commands are required to build a new package—or sometimes

several new packages. (One source RPM can build multiple binary RPMs.) The compilation process can take anywhere from a few seconds to several hours, depending on the size of the package and the speed of your computer. The result should be one or more new binary RPMs in /usr/src/ *distname*/RPMS/*arch*, where *distname* is a distribution-specific name (such as redhat on Red Hat or packages on SuSE) and *arch* is your CPU architecture (such as i386 or i586 for *x*86 or ppc for PowerPC). You can move these RPMs to any convenient location and install them just as you would any others.

> Source packages are also available for Debian systems, but aside from sites devoted to Debian and related distributions, Debian source packages are rare. The sites that do have these packages provide them in forms that typically install easily on appropriate Debian or related systems. For this reason, it's less likely that you'll rebuild a Debian package from source.

You can also recompile a package from a source tarball. This process is described later, in "Installing Programs from Source." This section also describes some of the potential pitfalls with compiling source code, whether from a tarball or using a source RPM.

Locating Another Version of the Problem Package

Frequently, the simplest way to fix a dependency problem or package conflict is to use a different version of the package you want to install. This could be a newer or older official version (4.2.3 rather than 4.4.7, say), or it might be the same official version but built for your distribution rather than for another distribution. Sites like RPM Find (http://www.rpmfind.net) or Debian's package listing (http://www.debian.org/distrib/packages) can be very useful in tracking down alternative versions of a package. Your own distribution's website or FTP site can also be a good place to locate packages as well.

> If the package you're trying to install requires newer libraries than you've got and you don't want to upgrade those libraries, an older version of the package may work with your existing libraries.

The main problem with locating another version of the package is that sometimes you really need the version that's not installing correctly. It might have features that you need, or it might fix important bugs. On occasion, other versions might not be available, or you might be unable to locate another version of the package in your preferred package format.

Startup Script Problems

One particularly common problem when trying to install servers from one distribution in another is in getting SysV startup scripts working. Although most major Linux distributions use SysV startup scripts, these scripts are not always transportable across distributions. Different distributions frequently implement support routines in unique ways, so these scripts may be incompatible.

The result is that the server you installed may not start up, even if the links to the startup scripts are correct. Possible workarounds include modifying the startup script that came with the server, building a new script based on another one from your distribution, and starting the server through a local startup script like `/etc/rc.d/rc.local` or `/etc/rc.d/boot.local`. Chapter 6, "The Boot Process and Scripts," describes startup scripts in more detail.

> **NOTE** Startup script problems affect only servers and other programs that are started automatically when the computer boots; they don't affect typical user applications or libraries.

Installing Programs from Source

Sometimes the package tools just don't do quite what you need. Perhaps the software you want is particularly obscure and is available only in source code. Perhaps you want to use the original software without any patches applied by a distribution maintainer. Perhaps you need to apply *your own* changes to the software. In any of these cases, the usual solution is to obtain the program's source code and compile it locally. To do so, you must know how to obtain and unpack the software, how to configure it, how to compile it, how to install it, and perhaps how to uninstall it.

> **NOTE** Compiling software yourself has its advantages and disadvantages. It enables you to optimize the software for your system and it enables you to change the software as you see fit or ensure that you're running the author's original software. On the downside, compiling software from source can take a long time (particularly if something goes wrong), and it bypasses your distribution's package system, thus eliminating its advantages.

Obtaining and Extracting Software

The first step in compiling software from source is to obtain it and extract the source code on your system. Most programs for Linux are available in source code form, but to obtain it, you must locate it. Unfortunately, there is no single repository for all open-source software, although several sites do host lots of it. The most important of these are SourceForge (`http://sourceforge.net`) and freshmeat (`http://freshmeat.net`). You can find many open-source projects hosted at these sites; try using their search systems to find ones that are of interest to you. Of course, you may already know what software you want to compile, in which case you may know its Web address or could at least search for it using a conventional Web search engine.

If you have an RPM source file, you can extract the pristine source code using `rpm2cpio`, as described earlier in "Extracting Data from RPMs." Most source RPMs provide the original, unmodified source code along with one or more patch files. You can discard the patch files (or apply them, if you like) and use the source tarball from the source RPM.

In any event, most Linux source code is distributed in the form of a tarball—an archive file created with `tar` and compressed with `gzip` or `bzip2`. Such files have filename extensions of `.tgz`, `.tar.gz`, `.tbz`, or `.tar.bz2`. The first two of these denote tarballs that use `gzip` compression, while the last two indicate `bzip2` compression.

Traditionally, you extract source code into the `/usr/src` directory, which holds subdirectories for different software packages. Ordinarily, only `root` may write to this directory, though, so you might prefer to do this job in an ordinary user's home directory. (An ordinary user can build most system software, which is safer than doing so as `root`, although the software must normally be installed by `root`.) To actually extract the software, you use the `tar` program, which is described more fully in Chapter 8, "Administering the System." Before extracting any files, I recommend you examine the contents of the tarball:

```
$ tar tvzf ~/someprog-1.2.3.tar.gz | less
```

This command creates a listing of the files in the tarball and pipes that listing through the `less` pager, enabling you to readily view the file's contents. Typically, source packages contain a single directory named after the program (such as *someprog-1.2.3*). For small programs, this directory may contain all the source code files, but larger programs typically include a series of subdirectories. A few programs don't create a main program subdirectory (such as *someprog-1.2.3*) at all; they contain nothing but the main source files. With them, you must be sure to create an appropriate subdirectory yourself and extract the files into that subdirectory by running subsequent `tar` commands from that directory.

The basic `tar` command to extract a source code tarball looks like this:

```
$ tar xvzf ~/someprog-1.2.3.tar.gz
```

This example extracts the *someprog-1.2.3.tar.gz* archive located in your home directory. You should change this filename to whatever is appropriate for your package, of course. The result should be the creation of the files and directories you discovered when you reviewed the package file's contents.

Both of these examples assume the tarball was compressed with `gzip`. The `z` option in the `tvzf` and `xvzf` option blocks is a code that tells `tar` to use `gzip` compression. If the tarball uses `bzip2` compression, though, you should substitute `j` for `z`:

```
$ tar xvjf ~/someprog-1.2.3.tar.bz2
```

The `j` option tells `tar` to use `bzip2`. An alternative form for either compression tool is to use a pipe:

```
$ gunzip -c ~/someprog-1.2.3.tar.gz | tar xvf -
```

For `bzip2` archives, substitute `bunzip2` for `gunzip` in this command. This more complex command is a holdover from traditional Unix systems, which had simpler `tar` commands that didn't interface as well with `gunzip` and `bunzip2`. Some programs' documentation suggests using this method, but the simpler command utilizing the `z` or `j` option to `tar` works as well under Linux.

However you do it, at the end of this process you should have a directory corresponding to the software you want to compile. This directory should hold source code files (possibly in one or more subdirectories), documentation, and so on.

Configuring Software

Unfortunately, standardization in procedures for configuring and compiling software from source code is incomplete at best. Some very small programs skip the configuration procedure altogether. Most programs, though, are complex enough that they include some sort of utility to help you configure the program. These utilities detect your OS, CPU, and libraries in order to create a customized `Makefile` for your system. This file controls the build process (which is handled by the `make` utility), so customizing it automatically enables the package to create a binary with compiler optimizations for your CPU, among other things.

> You should always read the documentation before building a source package. The documentation may describe options to modify the way the package is built, provide information on bugs and packages upon which the program depends, and so on.

The most common configuration processes require you to type one of two commands:

```
$ ./configure
$ make config
```

You need only type one of these commands, but which one you type depends on the package. (Some use other commands, too.) Consult your documentation for details. Both systems work in a similar way—the script uses various common system utilities to detect your CPU type, your OS, and so on. Some configure scripts also provide options that enable you to activate or deactivate features for your programs—that is, compile-time options. For instance, you might include or exclude support for advanced network authentication tools. In any event, the configure script creates a customized `Makefile` or other configuration files. In principle, you should then be able to move on to building the source code.

Sometimes, though, you may need to hand-tweak the configuration even after the configure script has done its work. To do so, you typically edit a file called `Makefile` or `makefile`. This file contains build instructions for the `make` utility, which you use to actually compile the software. A full description of the format of the `Makefile` is beyond the scope of this book, but you should be able to make some simple changes with only a minimal understanding of its contents.

The most common types of simple `Makefile` changes relate to the locations of common libraries, header files, utilities, and the ultimate installation directory. These options are set as you'd set variables in many programming languages, with an equal sign:

```
prefix = /usr/local
```

This specific example sets the `prefix` variable to `/usr/local`. This option is a common one; it tells `make` where to install the software when the time comes to do so. The `/usr/local` directory tree is commonly used for locally compiled software, so this setting is quite popular in default `Makefile` configurations. You could change it if you needed to do so, though; for instance, you could set `prefix` to `/opt` if you wanted to install the software in the `/opt` directory tree.

Other options set variables that are used to access system tools or utilities. For instance, the software might need to know where the `gzip` program is located:

```
AMPLOT_COMPRESS = /bin/gzip
```

In most cases, the package's configure script gets these details right; however, if the compilation fails in such a way that it appears to be trying to run programs from the wrong location, you could try looking for and changing such assignments. You might also want to make a substitution of some type. For instance, a common assignment tells `make` what C compiler to use:

```
CC = gcc
```

Subsequent calls to the C compiler use the `CC` variable rather than the `gcc` name. If you want to use another C compiler, though, you can change this assignment to do the trick.

> **NOTE**
> Frequently, large packages have multiple subdirectories, each with its own `Makefile`. The master `Makefile` contains references to the `Makefile` in each subdirectory. You can set global options in the main `Makefile`, or set options for a single subcomponent in its own `Makefile`.

Compiling and Installing Software

Actually compiling software is usually a matter of typing just one command:

```
$ make
```

This command causes the `make` utility to read the `Makefile` and execute a series of commands specified in the `Makefile`. Typically, the bulk of these commands are calls to `gcc` (or another compiler), followed by operations to *link* the individual object files (with `.o` filename extensions) created by `gcc` together into a single program file. Sometimes, though, other operations are equally important. For instance, `make` could build help files in several formats from a single help source file.

Whatever the details, the end result of typing make is one or more compiled programs, along with various temporary files and perhaps ancillary files (such as help files). Some packages place the main program file in the root of the source tree, but others put executable files in a subdirectory. Typically the program files should be installed before they're used, although many programs can be run from wherever they are. Particularly small and simple programs frequently leave the installation up to you: You must copy the program file, as root, to a convenient location using cp or mv:

```
# cp someprog /usr/local/bin
```

The traditional location for locally compiled programs is /usr/local, and their binaries usually go in /usr/local/bin. Before you type such a command, though, you should read the documentation to be sure that it's necessary. Most programs include a special make target to copy all the package's files to appropriate locations:

```
# make install
```

This command is likely to install binary files, man pages, support files, and so on. As with a simple cp installation, you must run this command as root.

At this point, the software is installed and should be usable. For user programs, you can try running the software to verify that it's working. For servers, consult the documentation; you may need to launch the server via a super server, as described in Chapter 9, "Basic Networking," or launch it via a startup script.

Once you're convinced that everything's working, you may want to clean up a bit. Type **make clean** to have make go through and remove temporary files. This saves space but will slow things down if you need to go back and recompile the package for any reason. To save even more space, of course, you can delete the entire source code directory tree; however, if you do so you won't be able to uninstall the software as easily.

Real World Scenario

Tracking Down Compilation Problems

Unfortunately, compiling software from source code frequently doesn't go as smoothly as just described. Most problems are caused by missing libraries. Such problems are often caught by the configure script—it will fail with a message to the effect that a particular library couldn't be found. Note that what's really missing isn't the library file that's used by the executable but the *development* library, which contains *header files*—files that enable source code to refer to library routines. Frequently, these files are included in separate packages from the main library. For instance, you might have the somelib package installed, but to compile software that uses somelib, you must install the somelib-devel package. Try searching your distribution for development libraries if you run into this problem.

Sometimes missing libraries don't become apparent until you're running make. The compile will proceed normally until it runs into a reference to the missing library, at which point it hangs, often with a message about a nonexistent file. Try doing a Web search on that filename; perhaps that will give you a clue about what it is. Sometimes you'll discover the file lurking on your system. If this happens, you may be able to get the compile to succeed by editing the Makefile to refer to the correct location of this file. You'll need to figure out which variable to change, though. Sometimes the names are similar enough that you won't have problems, but other times you'll need intimate familiarity with the software to figure out what to change.

Another common compile-time problem is compiler incompatibilities. These are particularly common if you're using a very old or very new compiler, if you're using an unusual compiler, or if you're compiling very old software. Compiler incompatibilities frequently manifest themselves as compiler errors—claims that the program's syntax is incorrect, for instance. Library version incompatibilities can cause similar problems, although these can often look more like missing libraries. The usual solution for such problems is to upgrade or downgrade either the software you're compiling or the compiler or library that's causing the problems.

Uninstalling Locally Compiled Software

Sometimes you install software only to discover that it's not what you wanted, or perhaps you use it for a while and then no longer need it. When this happens, you may want to uninstall the software. One of the great advantages of package management systems is that the package manager tracks the files associated with each package, enabling easy uninstallation. When you build packages from source code, you lose the integration with the package manager, so you must find another way to uninstall the software.

> **NOTE**   Two exceptions exist to the lack of package manager integration when you build from source code. One exception is if you use the package manager's own tools to build from source code, such as the rpmbuild program. When you build a binary package and then install it using the package manager, you can uninstall this software quite easily. The second exception is the Gentoo Linux distribution, which uses a package manager that builds almost everything locally.

The simplest packages make no provisions to help uninstall software. If you use such a package, you must manually track down and delete all the package's files in order to uninstall them. This task is tedious and can lead to error. Fortunately, most modern source packages include a special uninstall target to make. You use it much as you use the install target, as root:

```
# make uninstall
```

The result is that the make utility does all the dirty work for you, going through and deleting all the files it originally installed.

Managing Shared Libraries

Most Linux software relies heavily on *shared libraries.* The preceding sections have described some of the problems that can arise in managing the shared library packages—if a library isn't installed or is the wrong version, you may have problems installing a package. Library management goes beyond merely configuring them, though. To understand this, you must first understand a few library principles. You can then move on to setting the library path and using commands that manage libraries.

Library Principles

The idea behind a library is to simplify programmers' lives by providing commonly used program fragments. For instance, one of the most important libraries is the *C library (libc),* which provides many of the higher-level features associated with the C programming language. Another common type of library is associated with GUIs. These libraries are often called *widget sets* because they provide the on-screen *widgets* used by programs—buttons and scroll bars and menu bars and so on. The GIMP Tool Kit (GTK+) and Qt are the most popular Linux widget sets, and both ship largely as libraries. Libraries are chosen by programmers, not by users; you usually can't easily substitute one library for another. (The main exceptions are minor version upgrades.)

> Linux uses the *GNU C library (glibc)* version of the C library. Package manager dependencies and other library references are to glibc specifically. As of glibc 2.3.4, for historical reasons the main glibc file is usually called `/lib/libc.so.6`, but this file is sometimes a symbolic link to a file of another name, such as `/lib/libc-2.3.4.so`.

In principle, the routines in a library can be linked into a program's main file, just like all the object code files created by the compiler. This approach, however, has certain problems:

- The resulting program file is huge. This means that it takes up a lot of disk space and it consumes a lot of RAM when loaded.
- If multiple programs use the library, as is common, the program size issue is multiplied several times.
- The program can't take advantage of improvements in the library without recompiling (or at least relinking) the program.

For these reasons, most programs use their libraries as shared libraries (aka *dynamic libraries*). In this form, the main program executable omits most of the library routines. Instead, the executable includes references to shared library files, which can then be loaded along with the main program file. This approach helps keep program file size down, enables sharing of the memory consumed by libraries across programs, and enables programs to take advantage of improvements in libraries simply by upgrading the library.

> **NOTE** Linux shared libraries are similar to the dynamic link libraries (DLLs) of Windows. Windows DLLs are usually identified by .DLL filename extensions, but in Linux, shared libraries usually have .so or .so.*version* extensions, where *version* is a version number. (.so stands for "shared object.") Linux *static libraries* (used by linkers for inclusion in programs when dynamic libraries aren't to be used) have .a filename extensions.

On the downside, shared libraries can degrade program load time slightly if the library isn't already in use by another program, and they can create software management complications:

- Shared library changes can be incompatible with some or all programs that use the library. Linux uses library numbering schemes to enable you to keep multiple versions of a library installed at once. Upgrades that shouldn't cause problems can overwrite older versions, whereas major upgrades get installed side-by-side with their older counterparts. This approach minimizes the chance of problems, but sometimes changes that *shouldn't* cause problems *do* cause them.

- Programs must be able to locate shared libraries. This task requires adjusting configuration files and environment variables. If it's done wrong, or if a program overrides the defaults and looks in the wrong place, the result is usually that the program won't run at all.

- The number of libraries for Linux has risen dramatically over time. When they're used in shared form, the result can be a tangled mess of package dependencies, particularly if you use programs that rely on many or obscure libraries. In most cases, this issue boils down to a package problem that can be handled by your package management tools.

- If an important shared library becomes inaccessible because it was accidentally overwritten, because of a disk error or for any other reason, the result can be severe system problems. In a worst-case scenario, the system might not even boot.

In most cases, these drawbacks are manageable and are much less important than the problems associated with using static libraries. Thus, dynamic libraries are very popular.

> **NOTE** Static libraries are sometimes used by developers who create programs using particularly odd, outdated, or otherwise exotic libraries. This enables them to distribute their binary packages without requiring users to obtain and install their oddball libraries. Likewise, static libraries are sometimes used on small emergency systems, which don't have enough programs installed to make the advantages of shared libraries worth pursuing.

Locating Library Files

The major administrative challenge of handling shared libraries is in enabling programs to locate their shared libraries. Binary program files can point to libraries either by name alone (as in libc.so.6) or by providing a complete path (as in /lib/libc.so.6). In the first case, you must

configure a library path—a set of directories in which programs should search for libraries. This can be done both through a global configuration file and through an environment variable. If a static path to a library is wrong, you must find a way to correct the problem. In all of these cases, after making a change, you may need to use a special command to get the system to recognize the change, as described later in "Library Management Commands."

Setting the Path Systemwide

The first way to set the library path is to edit the `/etc/ld.so.conf` file. This file consists of a series of lines, each of which lists one directory in which shared library files may be found. Typically, this file lists between half a dozen and a couple dozen directories. Some distributions have an additional type of line in this file. These lines begin with the `include` directive; they list files that are to be included as if they were part of the main file. For instance, Fedora 3's `ld.so.conf` begins with this line:

```
include ld.so.conf.d/*.conf
```

This line tells the system to load all the files in `/etc/ld.so.conf.d` whose names end in `.conf` as if they were part of the main `/etc/ld.so.conf` file. This mechanism enables package maintainers to add their unique library directories to the search list by placing a `.conf` file in the appropriate directory.

A mechanism with a similar goal but different details is used by some other distributions, such as Gentoo. With these distributions, the `env-update` utility reads files in `/etc/env.d` to create the final form of several `/etc` configuration files, including `/etc/ld.so.conf`. In particular, the `LDPATH` variables in these files are read and their values make up the lines in `ld.so.conf`. Thus, to change `ld.so.conf` in Gentoo or other distributions that use this mechanism, you should add or edit files in `/etc/env.d` and then type **env-update** to do the job.

Generally speaking, there's seldom a need to change the library path system-wide. Library package files usually install themselves in directories that are already on the path or add their paths automatically. The main reason to make such changes would be if you installed a library package, or a program that creates its own libraries, in an unusual location via a mechanism other than your distribution's main package utility. For instance, you might compile a library from source code and then need to update your library path in this way.

After you change your library path, you must use `ldconfig` to have your programs use the new path, as described later in "Library Management Commands."

> In addition to the directories specified in `/etc/ld.so.conf`, Linux refers to the trusted library directories, `/lib` and `/usr/lib`. These directories are always on the library path, even if they aren't listed in `ld.so.conf`.

Temporarily Changing the Path

Sometimes changing the path permanently and globally is unnecessary and even inappropriate. For instance, you might want to test the effect of a new library before using it for all your programs. To do so, you could install the shared libraries in an unusual location and then set the

LD_LIBRARY_PATH environment variable. This environment variable specifies additional directories the system is to search for libraries.

> **NOTE**   Chapter 6 describes environment variables in more detail.

To set the LD_LIBRARY_PATH environment variable using the bash shell, you can type a command like this:

```
$ export LD_LIBRARY_PATH=/usr/local/testlib:/opt/newlib
```

This line adds two directories, /usr/local/testlib and /opt/newlib, to the search path. You can specify as few or as many directories as you like, separated by colons. In any event, these directories are added to the *start* of the search path, which means that they take precedence over other directories. This fact is handy when testing replacement libraries, but it can also cause problems if users manage to set this environment variable inappropriately.

You can set this environment variable permanently in a user's shell startup script files, as described in Chapter 6. Doing so means that the user will *always* use the specified library paths in addition to the normal system paths. In principle, you could set the LD_LIBRARY_PATH globally; however, using /etc/ld.so.conf is the preferred method of effecting global changes to the library path.

Unlike other library path changes, this one does not require that you run ldconfig for it to take effect.

Correcting Problems

Library path problems usually manifest as an inability of a program to locate a library. If you launch it from a shell, you'll see an error message like this:

```
$ gimp
gimp: error while loading shared libraries: libXinerama.so.1: cannot➥
 open shared object file: No such file or directory
```

This message indicates that the system couldn't find the libXinerama.so.1 library file. The usual cause of such problems is that the library isn't installed, so you should look for it using commands such as find (described in Chapter 4, "Managing Files and Filesystems"). If the file simply isn't installed, try to track down the package to which it should belong (a Web search can work wonders in this task) and install it.

If, on the other hand, the library file *is* available, you may need to add its directory globally or to the LD_LIBRARY_PATH. Sometimes the library's path is hard-coded in the program's binary file. (You can discover this using ldd, as described shortly, in "Library Management Commands.") When this happens, you may need to create a symbolic link from the location of the library on your system to the location the program expects. A similar problem can occur when the program expects a library to have one name but it has another name on your system. For instance, the program might link to biglib.so.5 but your system has biglib.so.5.2

installed. Minor version number changes like this are usually inconsequential, so creating a symbolic link will correct the problem:

```
# ln -s biglib.so.5.2 biglib.so.5
```

You must type this command as `root` in the directory in which the library resides. You must then run `ldconfig`, as described in the next section.

Library Management Commands

Linux provides a pair of commands that you're likely to use for library management. The `ldd` program displays the shared library dependencies of a program—that is, it lists the shared libraries that a program uses. The `ldconfig` program updates caches and links used by the system for locating libraries—that is, it reads `/etc/ld.so.conf` and implements any changes in that file or in the directories to which it refers. Both of these tools are invaluable in managing libraries.

Displaying Shared Library Dependencies

If you run into programs that won't launch because of missing libraries, the first step is to check which libraries the program file uses. You can do this with the `ldd` command:

```
$ ldd /bin/ls
        librt.so.1 => /lib/librt.so.1 (0x0000002a9566c000)
        libncurses.so.5 => /lib/libncurses.so.5 (0x0000002a95784000)
        libacl.so.1 => /lib/libacl.so.1 (0x0000002a958ea000)
        libc.so.6 => /lib/libc.so.6 (0x0000002a959f1000)
        libpthread.so.0 => /lib/libpthread.so.0 (0x0000002a95c17000)
        /lib64/ld-linux-x86-64.so.2 (0x0000002a95556000)
        libattr.so.1 => /lib/libattr.so.1 (0x0000002a95dad000)
```

Each line of output begins with a library name, such as `librt.so.1` or `libncurses.so.5`. If the library name doesn't contain a complete path, `ldd` attempts to find the true library and it displays the complete path following the `=>` symbol, as in `/lib/librt.so.1` or `/lib/libncurses.so.5`. You needn't be concerned about the long hexadecimal number following the complete path to the library file. The preceding example shows one library (`/lib64/ld-linux-x86-64.so.2`) that's referred to with a complete path in the executable file. It lacks the initial directory-less library name and `=>` symbol.

The `ldd` command accepts a few options. The most notable of these is probably `-v`, which displays a long list of version information following the main entry. This information might be helpful in tracking down which version of a library a program is actually using, in case you have multiple versions installed.

Keep in mind that libraries can themselves depend on other libraries. Thus, you can use `ldd` to discover what libraries are used by a library. Because of this potential for a dependency chain, it's possible that a program will fail to run even though all its libraries are present. When using `ldd` to track down problems, be sure to check the needs of all the libraries of the program, and all of the libraries used by the first tier of libraries, and so on until you've exhausted the chain.

The ldd utility can be run by ordinary users, as well as by root, although of course you must run it as root if you can't read the program file as an ordinary user.

Re-loading the Library Cache

Linux (or more precisely, the ld.so or ld-linux.so programs, which manage the loading of libraries) doesn't read /etc/ld.so.conf every time a program runs. Instead, the system relies on a cached list of directories and the files they contain, stored in binary format in /etc/ld.so.cache. This list is maintained in a format that's much more efficient than a plain-text list of files and directories. The drawback is that you must reload that cache every time you add or remove libraries. These additions and removals include both changing the contents of the library directories and adding or removing library directories.

The tool to do this job is called ldconfig. Ordinarily, it's called without any options:

```
# ldconfig
```

This program does, though, take options to modify its behavior:

Display verbose information   Ordinarily, ldconfig doesn't display any information as it works. The -v option causes the program to summarize the directories and files it's registering as it goes about its business.

Don't rebuild the cache   The -N option causes ldconfig to *not* perform its primary duty of updating the library cache. It will, though, update symbolic links to libraries, which is a secondary duty of this program.

Process only specified directories   The -n option causes ldconfig to update the links contained in the directories specified on the command line. The system won't examine the directories specified in /etc/ld.so.conf or the trusted directories (/lib and /usr/lib).

Don't update links.   The -X option is the opposite of -N; it causes ldconfig to update the cache but not manage links.

Use a new configuration file   You can change the configuration file from /etc/ld.so.conf by specifying it with the -f *conffile* option, where *conffile* is the file you want to use.

Use a new cache file   You can change the cache file that ldconfig creates by passing the -C *cachefile* option, where *cachefile* is the file you want to use.

Use a new root   The -r *dir* option tells ldconfig to treat *dir* as if it were the root (/) directory. This option is helpful when recovering a badly corrupted system or when installing a new OS.

Display current information   The -p option causes ldconfig to display the current cache—all the library directories and the libraries they contain.

Both RPM and Debian library packages typically run ldconfig automatically after installing or removing the package. The same thing happens as part of the installation process for many packages compiled from source. Thus, you may well be running ldconfig more than you realize in the process of software management. Still, you may need to run the program yourself if you manually modify your library configuration in any way.

Summary

Linux provides numerous tools to help you manage software. Most distributions are built around the RPM or Debian package systems, both of which enable installation, upgrade, and removal of software using a centralized package database to avoid conflicts and other problems that are common when no central package database exists. You can perform basic operations on individual files or, with the help of extra tools such as APT, keep your system synchronized with the outside world, automatically or semi-automatically updating all your software to the latest versions. In addition to these tools, you can use traditional Unix methods of software compilation and installation. This has the advantage of greater control on your part—you can set compile options for your system and even modify the source code. Installing software from source, though, eliminates the advantages of the RPM or Debian package database.

No matter how you install your software, you may need to deal with shared library management. These software components are necessary building blocks of large modern programs, and in the best of all possible worlds they operate entirely transparently. Sometimes, though, shared libraries need to be upgraded or the system configuration changed so that programs can find the libraries. When this happens, knowing about critical configuration files and commands can help you work around any difficulties.

Exam Essentials

Identify critical features of RPM and Debian package formats.   RPM and Debian packages store all files for a given package in a single file that also includes information on what other packages the software depends on. These systems maintain a database of installed packages and their associated files and dependencies.

Describe the tools used for managing RPMs.   The `rpm` program is the main tool for installing, upgrading, and uninstalling RPMs. This program accepts operations and options that tell it precisely what to do.

Describe the tools used for managing Debian packages.   The `dpkg` program installs or uninstalls a single package or a group of packages you specify. The `apt-get` utility retrieves programs from installation media or from the Internet for installation and can automatically upgrade your entire system. The `dselect` program serves as a menu-driven interface to `apt-get`, enabling you to select programs you want to install from a text-mode menu.

Summarize tools for extracting files and converting between package formats.   The `rpm2cpio` program can convert an RPM file to a `cpio` archive, enabling users of non-RPM systems to access files in an RPM. The `alien` utility can convert in any direction between Debian packages, RPMs, Stampede packages, and tarballs. This enables use of packages intended for one system on another.

Explain the process of compiling source code.   Source code compilation involves obtaining source code, extracting it into a source code directory, running a configure script, compiling the software with make, and installing the software. Details vary from one package to another, so you should always consult the documentation to learn precisely how to do it for any given program.

Identify the main advantages and disadvantages of compiling software from source code. Compiling software yourself enables you to ensure that you're using the author's original version of the program, to optimize the software for your system, and to modify the source code yourself. The main drawbacks are the extra time involved in the process and the fact that you're then bypassing your distribution's package utilities and thus get no benefit from them for the package you're compiling.

Summarize the reasons for using shared libraries.   Shared libraries keep disk space and memory requirements manageable by placing code that's needed by many programs in separate files from the programs that use it, enabling one copy to be used multiple times. More generally, libraries are useful by enabling programmers to use basic "building blocks" that others have written without having to constantly reinvent code.

Describe methods available to change the library path.   The library path can be changed system-wide by editing the /etc/ld.so.conf file and then typing **ldconfig**. For temporary or per-user changes, directories may be added to the path by placing them in the LD_LIBRARY_PATH environment variable.

Review Questions

1. Which of the following is *not* an advantage of a source package over a binary package?

   A. A single source package can be used on multiple CPU architectures.

   B. By recompiling a source package, you can sometimes work around library incompatibilities.

   C. You can modify the code in a source package, altering the behavior of a program.

   D. Source packages can be installed more quickly than binary packages can.

2. Which is true of using both RPM and Debian package management systems on one computer?

   A. It's generally inadvisable because the two systems don't share installed file database information.

   B. It's impossible because their installed file databases conflict with one another.

   C. It causes no problems if you install important libraries once in each format.

   D. It's a common practice on Red Hat and Debian systems.

3. Which of the following statements is true about binary RPM packages that are built for a particular distribution?

   A. They can often be used on another RPM-based distribution for the same CPU architecture, but this isn't guaranteed.

   B. They may be used in another RPM-based distribution only when you set the `--convert-distrib` parameter to `rpm`.

   C. They may be used in another RPM-based distribution only after you convert the package with `alien`.

   D. They can be recompiled for an RPM-based distribution running on another type of CPU.

4. Which is true of source RPM packages?

   A. They consist of three files: an original source tarball, a patch file of changes, and a PGP signature indicating the authenticity of the package.

   B. They require programming knowledge to rebuild.

   C. They can sometimes be used to work around dependency problems with a binary package.

   D. They are necessary to compile software for RPM-based distributions.

5. Which of the following do RPM filenames conventionally include?

   A. Single-letter codes indicating Red Hat–certified build sites

   B. Build date information

   C. Version number and CPU architecture information

   D. The initials of the package's maintainer

6. An administrator types the following command on an RPM-based Linux distribution:

    `# `**`rpm -ivh megaprog.rpm`**

    What is the effect of this command?

    A. The `megaprog` package, if it's installed, is uninstalled from the computer.

    B. The `megaprog.rpm` package, if it exists, is valid, and is not already installed, is installed on the system.

    C. The `megaprog.rpm` source RPM package is compiled into a binary RPM for the computer.

    D. Nothing; `megaprog.rpm` is not a valid RPM filename, so `rpm` will refuse to operate on this file.

7. Which of the following commands will extract the contents of the `myfonts.rpm` file into the current directory?

    A. **`rpm2cpio myfonts.rpm | cpio -i --make-directories`**

    B. **`rpm2cpio myfonts.rpm > make-directories`**

    C. **`rpm -e myfonts.rpm`**

    D. **`alien --to-extract myfonts.rpm`**

8. To use `dpkg` to remove a package called `theprogram`, including its configuration files, which of the following commands would you issue?

    A. **`dpkg -P theprogram`**

    B. **`dpkg -p theprogram`**

    C. **`dpkg -r theprogram`**

    D. **`dpkg -r theprogram-1.2.3-4.deb`**

9. Which of the following describes a difference between `apt-get` and `dpkg`?

    A. `apt-get` provides a GUI interface to Debian package management; `dpkg` does not.

    B. `apt-get` can install tarballs in addition to Debian packages; `dpkg` cannot.

    C. `apt-get` can automatically retrieve and update programs from Internet sites; `dpkg` cannot.

    D. `apt-get` is provided only with the original Debian distribution, but `dpkg` comes with Debian and its derivatives.

10. Which of the following is true of an attempt to use a Debian package from one distribution on another Debian-derived distribution?

    A. It's unlikely to work because of library incompatibilities and divergent package-naming conventions.

    B. It's guaranteed to work because of Debian's strong package definition and enforcement of standards for startup scripts and file locations.

    C. It will work only when the distributions are built for different CPUs or when the `alien` package is already installed on the target system.

    D. It's likely to work because of the close relationship of Debian-based distributions, assuming the two distributions are for the same CPU architecture.

11. Which of the following is the default destination format when using `alien`?

    A. Tarball

    B. RPM

    C. Debian package

    D. Stampede package

12. How do you select packages for installation using `dselect`?

    A. You highlight the package in the selection list and press the plus (+) key.

    B. You right-click the package name with the mouse and select the Mark for Installation option.

    C. You highlight the package in the selection list and press the spacebar key.

    D. You click the package name with the mouse and pick the Install ➢ Package option from the main menu.

13. As `root`, you type **apt-get update** on a Debian system. What should be the effect of this command?

    A. None; `update` is an invalid option to `apt-get`.

    B. The APT utilities deliver information on the latest updates you've made to the APT Internet repositories, enabling you to share your changes with others.

    C. The APT utilities download all available upgrades for your installed programs and install them on your system.

    D. The APT utilities retrieve information on the latest packages available so that you may install them with subsequent `apt-get` commands.

14. Which of the following commands would you use to extract the contents of the `newprog-5.3.tbz` source code tarball? (Select all that apply.)

    A. **tar xvjf newprog-5.3.tbz**

    B. **bunzip2 -c newprog-5.3.tbz | tar xvf -**

    C. **tar xvzf newprog-5.3.tbz**

    D. **rpm2cpio newprog-5.3.tbz | cpio -i --make-directories**

15. Which of the following is *not* a good reason for installing software from source code?

    A. Installing software from source code enables you to optimize the software for your CPU.

    B. Installing software from source code helps maintain your RPM or Debian package database.

    C. Installing software from source code enables you to make changes to the source code.

    D. Installing software from source code enables you to tweak program options to your liking.

16. Which of the following steps should you always take before attempting to compile software from source code?

    A. Edit the `Makefile` to optimize the compilation for your CPU.

    B. Read the documentation provided with the source code.

    C. Type **make config** to configure the software for your computer.

    D. Ensure that the Qt and GTK+ libraries are installed for use by the software.

17. What methods might you use to uninstall a program you installed from source code without the help of a package manager? (Select all that apply.)

   A. Consult the Registry to learn what files are installed and delete those associated with the package.

   B. Type **make uninstall** in the original program source code directory.

   C. Type **rpm -e *packagename***, where *packagename* is the name of the package.

   D. Manually track down and delete all the files installed by the package.

18. What is the function of the ldd program?

   A. It displays information on the libraries used by a program or a library.

   B. It copies a file, optionally applying conversions to it.

   C. It causes the system to re-create the library cache files and renew library links.

   D. It flushes the LD_LIBRARY_PATH environment variable.

19. What is the preferred method of adding a directory to the library path for all users?

   A. Modify the LD_LIBRARY_PATH environment variable in a global shell script.

   B. Add the directory to the /etc/ld.so.conf file and then type **ldconfig**.

   C. Type **ldconfig */new/dir***, where */new/dir* is the directory you want to add.

   D. Create a symbolic link from that directory to one that's already on the library path.

20. You prefer the look of GTK+ widgets to Qt widgets, so you want to substitute the GTK+ libraries for the Qt libraries on your system. How would you do this?

   A. You must type **ldconfig --makesubs=qt,gtk**. This command substitutes the GTK+ libraries for the Qt libraries at load time.

   B. You must uninstall the Qt library packages and re-install the GTK+ packages with the --substitute=qt option to rpm or the --replace=qt option to dpkg.

   C. You must note the filenames of the Qt libraries, uninstall the packages, and create symbolic links from the Qt libraries to the GTK+ libraries.

   D. You can't easily do this; libraries cannot be arbitrarily exchanged for one another. You would need to rewrite all the Qt-using programs to use GTK+.

Answers to Review Questions

1. D. Because they must be compiled prior to installation, source packages require *more* time to install than binary packages do, contrary to option D's assertion. The other options all describe advantages of source packages over binary packages.

2. A. Package management systems don't share information, but neither do their databases actively conflict. Installing the same libraries using both systems would almost guarantee that the files served by both systems would conflict with one another. Actively using both RPM and Debian packages isn't common on any distribution, although it's possible with all of them.

3. A. RPMs are usually portable across distributions, but occasionally they contain incompatibilities. There is no `--convert-distrib` parameter to `rpm`, nor is `alien` used to convert from RPM format to RPM format. Binary packages can't be rebuilt for another CPU architecture, but source packages may be rebuilt for any supported architecture provided the source code doesn't rely on any CPU-specific features.

4. C. Some dependencies result from dynamically linking binaries to libraries at compile time and so they can be overcome by recompiling the software from a source RPM. Option A describes Debian source packages, not RPM packages. Recompiling a source RPM requires only issuing an appropriate command, although you must also have appropriate compilers and libraries installed. Source tarballs can also be used to compile software for RPM systems, although this results in none of RPM's advantages.

5. C. The package version number (as well as an RPM build number) and CPU architecture code (or `src` for source code or `noarch` for architecture-independent files) are included in most RPM package filenames. Red Hat does not provide certification for RPM maintainers. Build dates and package maintainers' names are stored in the RPM, but not in the filename. (Some distributions include a code for the distribution name in the RPM filename, but this is not a universal practice.)

6. B. The `-i` operation installs software, so option B is correct. (The `-v` and `-h` options cause a status display of the progress of the operation, which wasn't mentioned in the option.) Uninstallation is performed by the `-e` operation, and rebuilding source RPMs is done by the `--rebuild` operation (to either `rpm` or `rpmbuild`, depending on the RPM version). Although the filename `megaprog.rpm` is missing several conventional RPM filename components, the `rpm` utility doesn't use the filename as a package validity check, so option D is incorrect.

7. A. The `rpm2cpio` program extracts data from an RPM file and converts it into a `cpio` archive that's sent to standard output. Piping the results through `cpio` and using the `-i` and `--make-directories` options, as in option A, will extract those files to the current directory. Option B creates a `cpio` file called `make-directories` that contains the files from the RPM package. Option C will uninstall the package called `myfonts.rpm` (but not the `myfonts` package). The `alien` utility has no `--to-extract` target, so option D is invalid.

8. A. An uppercase `-P` invokes the purge operation, which completely removes a package and its configuration files. The lowercase `-p` causes `dpkg` to print information on the package's contents. The `-r` parameter removes a package but leaves configuration files behind. The final variant (option D) also specifies a complete filename, which isn't used for removing a package—you should specify only the shorter package name.

9. C. You can specify Debian package archive sites in /etc/apt/sources.list, and then you can type **apt-get update** and **apt-get upgrade** to quickly update a Debian system to the latest packages. GUI package management tools for Debian and related distributions exist, but they aren't apt-get. The alien program can convert an RPM file and install the converted package on a Debian system; dpkg and apt-get both come with all Debian-based distributions.

10. D. Systems that use Debian are based on the same core OS and so they share most components, making package transplants likely—but not certain—to succeed. Library incompatibilities *could* cause problems but aren't likely to, especially if you use recent packages and distributions. Although Debian has clearly defined key file locations, startup scripts, and so on, these can't guarantee success. Binary packages built for *different* CPUs are almost guaranteed *not* to work, although scripts or other non-binary packages most likely will work across CPU types.

11. C. The alien utility can convert to any of these formats, but if you omit the specification, it defaults to a Debian output package.

12. A. The dselect program is a text-based menuing interface to APT or other Debian package tools. Selecting packages to install involves highlighting them and then pressing the plus (+) key. The spacebar, as in option C, does not select packages for installation. Options B and D both describe an X-based GUI tool, but dselect is a text-mode program.

13. D. The update option to apt-get causes retrieval of new information, as described in option D. This option is perfectly valid, contrary to option A's assertion. The apt-get program doesn't permit you to upload information to the Internet repositories, so option B is incorrect. Option C describes the effect of the upgrade or dist-upgrade options, not the update option.

14. A, B. The package filename ends in .tbz, which means it's a tarball that's been compressed with bzip2. This tarball can be extracted by tar if you pass it the -j option, as in option A; or you can uncompress the tarball with bunzip2 and pipe the results through tar without any compression options, as in option B. Option C would work if the file were compressed with gzip; its -z option is appropriate for that compression tool. Option D simply will not work, although it's appropriate for extracting the contents of an RPM file without installing it using the RPM package system.

15. B. Your RPM or Debian package database is not used when you install software from source code, so option B is simply an incorrect statement. Options A, C, and D are all advantages of building software from source code.

16. B. Reading the documentation ensures that you'll be aware of any unique requirements or procedures for this software, thus increasing your chances of successfully compiling and installing it. Although you *can* tweak the Makefile, as described in option A, this isn't a requirement for compilation, and manual tweaks are often unnecessary. The **make config** procedure of option C is indeed necessary for *some* programs, but many others use other configuration procedures, so this precise action is not always required. The Qt and GTK+ libraries referred to in option D are necessary for some programs but not for all of them, so this step is not always required.

17. B, D. Many programs provide an `uninstall` target for `make`, so option B will often (but not always) work. When it doesn't, or if you no longer have the original source code files, you must remember or be able to figure out which files belong to the package and delete them, as described in option D. Linux has no Registry; that's a Windows concept, so option A is invalid. Option C would work if you'd installed the package using RPM, but this procedure is useless for packages compiled from source. (An exception would be if you compiled a source RPM file and then installed the resulting binary file, but the question indicates this wasn't the case.)

18. A. Option A correctly summarizes the function of `ldd`. Option B summarizes the function of `dd`, an unrelated program described in Chapter 8. Option C describes the function of `ldconfig`. Option D is pure fiction, although the `LD_LIBRARY_PATH` environment variable is real.

19. B. The `/etc/ld.so.conf` file holds the global library path, so editing it is the preferred approach. You must then type **`ldconfig`** to have the system update its library path cache. Although you can add a directory to the library path by altering the `LD_LIBRARY_PATH` environment variable globally, this approach is not the preferred one. Option C simply won't work. Option D also won't work, although linking individual library files would work. This method is not the preferred one for adding a whole directory, though.

20. D. Libraries are selected by programmers, not by users or system administrators. If you don't like the widgets provided by one library, you have few options. (Many widget sets do provide a great deal of configurability, though, so you may be able to work around the problem in other ways.) Options A and B describe completely fictitious options to `ldconfig`, `rpm`, and `dpkg`. Option C would not work; Qt-using programs would simply crash when they found GTK+ libraries in place of the Qt libraries they were expecting.

Chapter

3

Configuring Hardware

THE FOLLOWING LINUX PROFESSIONAL INSTITUTE OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **1.101.1 Configure fundamental BIOS settings (weight: 1)**
- ✓ **1.101.3 Configure modem and sound cards (weight: 1)**
- ✓ **1.101.4 Set up SCSI devices (weight: 1)**
- ✓ **1.101.5 Set up different PC expansion cards (weight: 3)**
- ✓ **1.101.6 Configure communication devices (weight: 1)**
- ✓ **1.101.7 Configure USB devices (weight: 1)**
- ✓ **1.102.1 Design hard disk layout (weight: 5)**
- ✓ **1.102.2 Install a boot manager (weight: 1)**
- ✓ **1.104.1 Create partitions and filesystems (weight: 3)**

All OSs run atop hardware, and this hardware influences how the OSs run. Most obviously, hardware can be fast or slow, reliable or unreliable. Somewhat more subtly, OSs provide various means of configuring and accessing the hardware—initializing RS-232 serial ports and partitioning hard disks, for instance. You must understand at least the basics of how Linux interacts with its hardware environment in order to effectively administer a Linux system, so this chapter presents this information.

This chapter begins with a look at the BIOS, which is the lowest-level software that runs on a computer. The BIOS starts the boot process and configures certain hardware devices. This chapter then moves on to expansion cards, modems, audio hardware, and USB devices. Finally, this chapter covers disk hardware in more detail, including SCSI hardware, disk partitioning, and maintaining a boot loader, which helps boot Linux.

Configuring the BIOS and Core Hardware

All computers ship with a set of core hardware—most obviously, a *central processing unit (CPU),* which does the bulk of the computational work, and *random access memory (RAM),* which holds data. Many additional basic features help glue everything together, though, and some of these can be configured both inside and outside of Linux. At the heart of much of this hardware is the *Basic Input/Output System (BIOS),* which provides configuration tools and initiates the OS booting process. You can use the BIOS to enable and disable key hardware components, but once Linux is booted, you may need to manage this hardware using Linux utilities. Key components managed by the BIOS (and, once it's booted, Linux) include interrupts, I/O addresses, DMA addresses, the real-time clock, and *Advanced Technology Attachment (ATA)* hard disk interfaces.

Understanding the Role of the BIOS

The BIOS is the firmware that initiates the process of booting an operating system on a computer. It resides on the motherboard in ROM, typically in an *electronically erasable programmable read-only memory (EEPROM)*, aka flash memory. When you turn on a computer, the BIOS performs a *power-on self-test (POST)*, initializes hardware to a known operational state, loads the boot loader from the boot device (typically the first hard disk), and passes control to the boot loader, which in turn loads the OS.

Historically, a further purpose of the BIOS was to provide fundamental input/output (I/O) services to the operating system and application programs, insulating them from hardware changes. While the Linux kernel uses the BIOS to collect information about the hardware, once running, it doesn't use BIOS services for I/O. Having said that, Linux system administrators require a basic understanding of the BIOS because of the key role it plays in configuring hardware and in booting.

While BIOS implementations vary from manufacturer to manufacturer, most provide an interactive facility to configure them. Typically, you enter this setup tool by pressing the Delete, F1, or F2 key early in the boot sequence. (Consult your motherboard manual or look for on-screen prompts for details.) Figure 3.1 shows a typical BIOS setup main screen. You can use the arrow keys, the Enter key, and so on to move around the BIOS options and adjust them. Computers usually come delivered with reasonable BIOS defaults, but you may need to adjust them if you add new hardware. For instance, you might want to disable an onboard RS-232 port to make way for an internal modem that you add.

IRQs

An *interrupt request (IRQ)*, or an interrupt, is a signal sent to the CPU instructing it to suspend its current activity and to handle some external event such as keyboard input. On the $x86$ platform, IRQs are numbered from 0 to 15. Some are reserved for specific purposes such as the keyboard and the real-time clock, others have common uses (and are sometimes overused) but may be reassigned, and some are left available for extra devices that may be added to the system. Table 3.1 lists the IRQs and their common purposes.

F I G U R E   3 . 1   The BIOS setup screen provides features related to low-level hardware configuration.
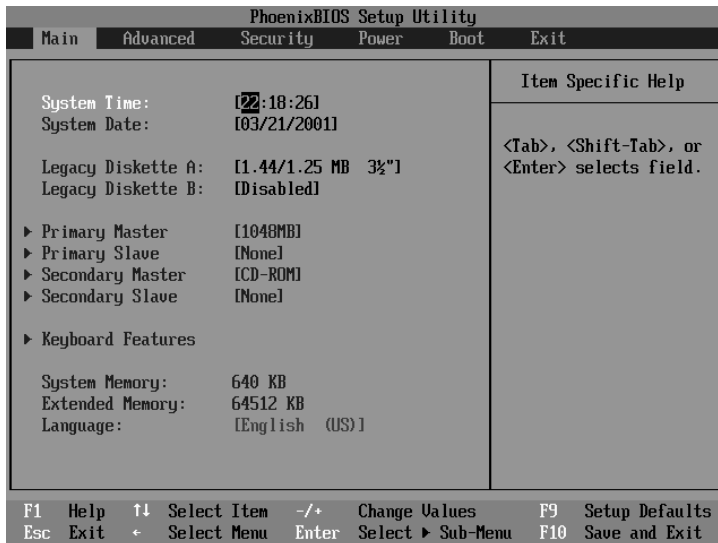
T A B L E   3 . 1   IRQs and Their Common Uses

IRQ	Typical Use	Notes
0	System timer	Reserved for internal use.
1	Keyboard	Reserved for keyboard use only.
2	Cascade for IRQs 8–15	The original *x*86 IRQ-handling circuit can manage just 8 IRQs; 2 are tied together to handle 16 IRQs, but IRQ 2 must be used to handle IRQs 8–15.
3	Second RS-232 serial port (COM2: in Windows)	May also be shared by a fourth RS-232 serial port.
4	First RS-232 serial port (COM1: in Windows)	May also be shared by a third RS-232 serial port.
5	Sound card or second parallel port (LPT2: in Windows)	
6	Floppy disk controller	Reserved for the first floppy disk controller.
7	First parallel port (LPT1: in Windows)	
8	Real-time clock	Reserved for system clock use only.
9	Open interrupt	
10	Open interrupt	
11	Open interrupt	
12	PS/2 mouse	
13	Math coprocessor	Reserved for internal use.
14	Primary ATA controller	The controller for ATA devices such as hard drives; typically /dev/hda and /dev/hdb under Linux.
15	Secondary ATA controller	The controller for more ATA devices; typically /dev/hdc and /dev/hdd under Linux.

The original *Industry Standard Architecture (ISA)* bus design makes sharing an interrupt between two devices tricky. Ideally, every ISA device should have its own IRQ. The more recent *Peripheral Component Interconnect (PCI)* bus makes sharing interrupts a bit easier, so PCI devices do frequently end up sharing an IRQ. The ISA bus has become rare on computers made since 2001 or so, but it's quite common on older computers. Even some computers that lack physical ISA slots may use ISA internally to manage some devices, such as RS-232 serial ports and parallel ports.

> **WARNING**
>
> IRQ 5 is a common source of interrupt conflicts because it is the default value for sound cards as well as for second parallel ports.

Once a Linux system is running, you can explore what IRQs are being used for various purposes by examining the contents of the /proc/interrupts file. A common way to do this is with the use of the cat command:

```
# cat /proc/interrupts
          CPU0
   0:    1132592       XT-PIC  timer
   1:         12       XT-PIC  keyboard
   2:          0       XT-PIC  cascade
   3:      39035       XT-PIC  orinoco_cs
   8:          4       XT-PIC  rtc
  11:        107       XT-PIC  eth0, Intel 82801DB-ICH4
  12:          9       XT-PIC  PS/2 Mouse
  14:          0       XT-PIC  ide0
  15:      26071       XT-PIC  ide1
NMI:          0
ERR:          0
```

> **NOTE**
>
> The /proc filesystem is a virtual filesystem—it doesn't refer to actual files on a hard disk, but to data that's convenient to represent using a filesystem. The files in /proc provide information about the hardware, running processes, and so on. Many Linux utilities use /proc behind the scenes, or you can directly access these files using utilities like cat, which copies the data to the screen when given just one argument.

This output shows the names of the drivers that are using each IRQ. Some of these driver names are easy to interpret, such as keyboard. Others are more puzzling, such as orinoco_cs (it's a driver for a wireless networking card). If the purpose of a driver isn't obvious, try doing a Web search on it; chances are you'll find a relevant hit fairly easily.

> **WARNING**   The /proc/interrupts file lists IRQs that are in use by Linux; however, Linux doesn't begin using an IRQ until the relevant driver is loaded. This may not happen until you try to use the hardware. Thus, the /proc/interrupts list may not show all the interrupts that are actually configured on your system. For instance, the preceding example shows nothing for IRQ 6, which is reserved for the floppy disk, because it hadn't been used prior to viewing the file. If the floppy disk were used and /proc/interrupts viewed again, an entry for IRQ 6 and the floppy driver would appear.

If your system suffers from IRQ conflicts, you must reconfigure one or more devices to use different IRQs. This topic is described shortly, in "Configuring Expansion Cards."

I/O Addresses

I/O addresses (also referred to as I/O ports) are unique locations in memory that are reserved for communications between the CPU and specific physical hardware devices. Like IRQs, I/O addresses are commonly associated with specific devices and should not ordinarily be shared. Table 3.2 shows a listing of some Linux device filenames along with the equivalent names in Windows, as well as the common IRQ and I/O Address settings.

TABLE 3.2   Common Linux Devices

Linux Device	Windows Name	Typical IRQ	I/O Address
/dev/ttyS0	COM1	4	0x03f8
/dev/ttyS1	COM2	3	0x02f8
/dev/ttyS2	COM3	4	0x03e8
/dev/ttyS3	COM4	3	0x02e8
/dev/lp0	LPT1	7	0x0378-0x037f
/dev/lp1	LPT2	5	0x0278-0x027f
/dev/fd0	A:	6	0x03f0-0x03f7
/dev/fd1	B:	6	0x0370-0x0377

Although the use is deprecated, older systems sometimes use /dev/cua*x* (where *x* is a number from 0 and up) to indicate an RS-232 serial device. Thus, /dev/ttyS0 and /dev/cua0 would refer to the same physical device.

Once a Linux system is running, you can explore what I/O addresses are being used by examining the contents of the /proc/ioports file. A common way to do this is with the use of the cat command.

```
# cat /proc/ioports
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
0070-007f : rtc
00f0-00ff : fpu
0170-0177 : ide1
01f0-01f7 : ide0
0376-0376 : ide1
03c0-03df : vga+
03f6-03f6 : ide0
```

As with IRQs, if your system suffers from I/O port conflicts, you must reconfigure one or more devices, as described in "Configuring Expansion Cards." In practice, such conflicts are rarer than IRQ conflicts.

DMA Addresses

Direct memory addressing (DMA) is an alternative method of communication to I/O ports. Rather than have the CPU mediate the transfer of data between a device and memory, DMA permits the device to transfer data directly, without the CPU's attention. The result can be lower CPU requirements for I/O activity, which can improve overall system performance.

To support DMA, the *x*86 architecture implements several DMA channels, each of which can be used by a particular device. To learn what DMA channels are in use on your system, examine the /proc/dma file:

```
$ cat /proc/dma
 3: parport0
 4: cascade
```

This output indicates that DMA channels 3 and 4 are in use. As with IRQs and I/O ports, DMA addresses should not normally be shared. In practice, DMA address conflicts are rarer than IRQ conflicts, so chances are you won't run into problems. If you do, consult the upcoming section, "Configuring Expansion Cards."

The Real-Time Clock

The *real-time clock (RTC)* is the battery powered clock used to keep track of time when the system is powered off. It's also known as the hardware clock or the *complementary metal oxide semiconductor (CMOS)* clock. Usually, the BIOS setup utility has a facility for setting the RTC.

Once running, the Linux kernel doesn't use the RTC directly to keep track of time. Instead, it uses the timer interrupt to maintain a software counter known as the *system clock* or the *software clock,* which gets its initial value from the RTC at startup.

> The date command displays the date and time represented by the system clock, not the RTC. You can read and write the RTC from within Linux using hwclock. The commands are covered in Chapter 8, "Administering the System."

Boot Disks and Geometry Settings

Most BIOSes allow you to choose the order of the devices from which to boot, falling back to the second entry if the first fails, the third entry if the second fails, and so on. A common order is the first floppy (known as A: in DOS and Windows), followed by the CD-ROM, followed by the first hard disk. With this configuration, the system will attempt to boot from each device in turn until one works. If all the devices fail to boot, the BIOS displays an error message.

While this is a common boot sequence, it has its problems. Specifically, if somebody accidentally leaves a floppy disk in the drive, this can prevent the system from booting. Worse, some (mostly old) viruses are transmitted by floppy disks' boot sectors, so this method can result in viral infection. Using removable disks as the default boot media also opens the door to intruders who have physical access to the computer; they need only reboot with a bootable floppy disk or CD-ROM to gain complete control of your system. For these reasons, it's better to make the first hard disk the only boot device. (You must change this configuration when installing Linux or using an emergency boot disk for maintenance.) Some BIOSes (most commonly on notebooks) make temporary changes easier by providing a special key to allow a one-time change to the boot sequence. On most other computers, to change the boot sequence, you must locate the appropriate BIOS option, change it, and reboot the computer. It's usually located in an Advanced menu, so look there.

Another disk option is the one for detection of disk devices. Figure 3.1 shows three disk devices: the A: floppy disk (/dev/fd0 under Linux), a 1048MB primary master hard disk, and a CD-ROM drive as the secondary master. In most cases, the BIOS detects and configures hard disks and CD-ROM drives correctly. You may need to tell it what sort of floppy disk you've got, though. Also, in rare circumstances, you must tell the BIOS about the hard disk's *cylinder/head/sector (CHS) geometry*.

The CHS geometry is a holdover from the early days of the *x*86 architecture. Figure 3.2 shows the traditional hard disk layout, which consists of a fixed number of read/write heads that can move across the disk surfaces (or platters). As the disk spins, each head marks out a circular track on its platter; these tracks collectively make up a cylinder. Each track is broken down into a series of sectors. Thus, any sector on a hard disk can be uniquely identified by three numbers: a cylinder number,

a head number, and a sector number. The *x*86 BIOS was designed to use this three-number CHS identification code. One consequence of this configuration is that the BIOS must know how many cylinders, heads, and sectors the disk has. Modern hard disks relay this information to the BIOS automatically, but for compatibility with the earliest hard disks, BIOSes still enable you to set these values manually.

> The BIOS will detect only certain types of disks. Of particular importance, SCSI disks and serial ATA disks won't appear in the main BIOS disk detection screen. These disks are handled by supplementary BIOSes associated with the controllers for these devices. Some BIOSes do provide explicit options to add SCSI devices into the boot sequence, though, so you can give priority to either ATA or SCSI devices. For those without these options, SCSI disks generally take second seat to ATA disks.

CHS geometry, unfortunately, has its problems. For one thing, all but the earliest hard disks use variable numbers of sectors per cylinder—modern disks squeeze more sectors onto outer tracks than inner ones, fitting more data on each disk. Thus, the CHS geometry presented to the BIOS by the hard disk is a convenient lie. Worse, because of limits on the numbers in the BIOS and in the ATA hard disk interface, plain CHS geometry tops out at 540MB, which is puny by today's standards. Various patches, such as CHS geometry translation, can be used to expand the limit to about 8GB. Today, though, the preference is to use *logical block addressing (LBA)* mode. (Some sources use the expansion *linear block addressing* for this acronym.) In this mode, a single unique number is assigned to each sector on the disk, and the disk's firmware is smart enough to read from the correct head and cylinder when given this sector number. Modern BIOSes typically provide an option to use LBA mode, CHS translation mode, or possibly some other modes with large disks. In most cases, LBA mode is the best choice. If you must retrieve data from very old disks, though, you might need to change this option.

FIGURE 3.2 Hard disks are built from platters, each of which is broken into tracks, which are broken into sectors.
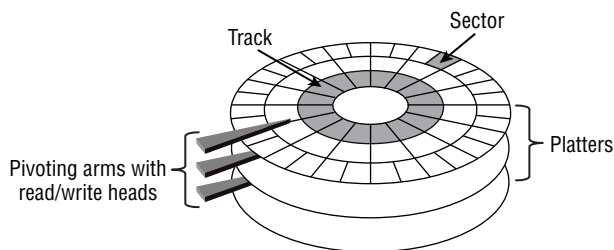
WARNING Because of variability in how different BIOSes handle CHS translation, moving disks between computers can result in problems because of mismatched CHS geometries claimed in disk structures and by the BIOS. Linux is usually smart enough to work around such problems, but you may see some odd error messages in disk utilities like fdisk. If you see messages about inconsistent CHS geometries, proceed with caution when using low-level disk utilities lest you create an inconsistent partition table that could cause problems, particularly in OSs that are less robust than Linux on this score.

Configuring Expansion Cards

Many hardware devices require configuration—you must set the IRQ, I/O port, and DMA addresses used by the device. (Not all devices use all three resources, though.) Through the mid-1990s, this process involved tedious changes to jumpers on the hardware. Today, though, most options can be configured through software. To this end, you must know something about the methods used to configure both ISA cards and PCI cards.

NOTE Even devices that are built into the motherboard are configured through the same means used to configure ISA or PCI cards.

Configuring ISA Devices

Very old ISA cards are configured through jumpers on the cards. For instance, you might set one jumper to adjust the IRQ used by the device and a second jumper to adjust its I/O port. These settings are very device specific, so you must consult the documentation for your particular card. Unfortunately, this process is error prone and tedious. For this reason, in the mid-1990s a new configuration method for ISA cards was developed. Known as *Plug and Play (PnP)*, the idea was to create jumperless ISA cards that could be automatically configured by the OS's drivers. In practice, ISA PnP devices are usually easier to configure and manage than are jumpered ISA devices, but ISA PnP devices do still require some configuration. Two mechanisms exist to do this: in-kernel support in 2.4.*x* and later kernels and the isapnp utilities that are used in earlier kernels and optionally in later ones.

Using In-Kernel PnP Support

The 2.4.*x* kernels added direct kernel support for the ISA PnP protocols. This support enables kernel drivers to directly configure ISA PnP devices. To use it, you must locate and activate the appropriate support. Figure 3.3 shows the relevant menu in a 2.6.10 kernel configuration tool; it's part of the Device Drivers ➢ Plug and Play Support area. Enable both the main Plug and Play

Support option and all the relevant protocols—if in doubt, activate all of them. (Chapter 6, "The Boot Process and Scripts," describes the basics of compiling your own kernel.) Modern Linux distributions ship with these options enabled by default, so you shouldn't need to adjust it if you use the standard kernel.

Once the in-kernel PnP support option is activated, the kernel will attempt to activate and configure PnP devices using the drivers' default preferences, driver options, and the needs of other devices as guides to what is possible. In most cases, this works well. If you need to adjust a configuration, you can do so by passing options to the modules for the relevant drivers. (Chapter 6 describes how to do this.)

Using Non-Kernel Utilities

If you're using an old Linux distribution with a pre-2.4.*x* kernel, you must use an extra package to manage ISA PnP devices. Indeed, even 2.4.*x* and later kernels permit use of this mechanism if you prefer. The package as a whole is called `isapnp`, and it includes two important utilities: `pnpdump` and `isapnp`.

Typically, you begin configuring ISA PnP devices by using `pnpdump`, which retrieves the current configuration of ISA PnP devices:

```
# pnpdump > isapnp.conf
```

FIGURE 3.3   Linux's in-kernel ISA PnP support requires you to activate the appropriate options when you compile the kernel.
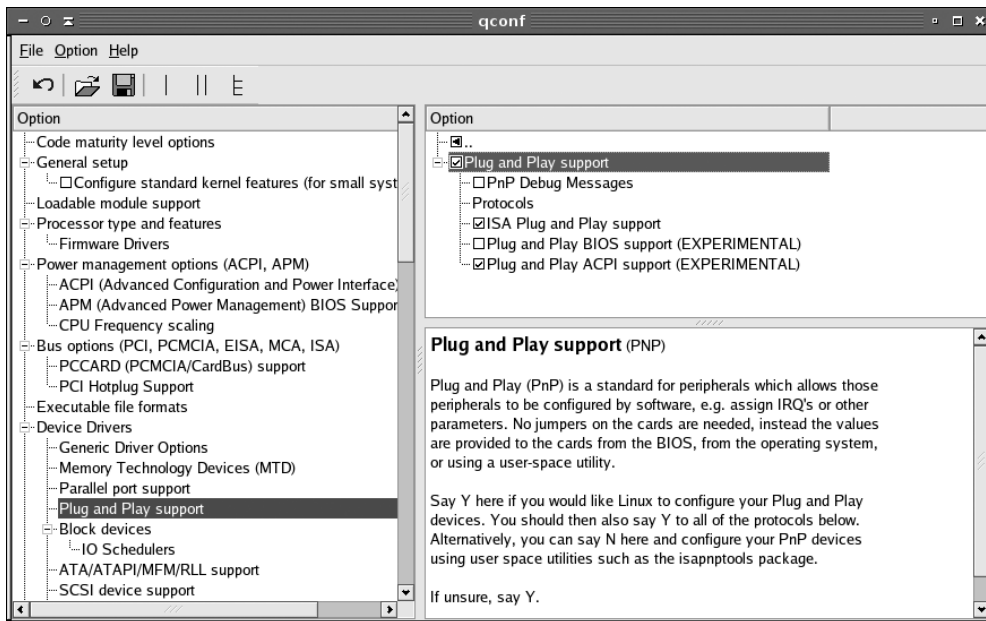
This command dumps the PnP configuration to the `isapnp.conf` file. (The `isapnp` utility frequently uses a file called `/etc/isapnp.conf` as its input.) Load the `isapnp.conf` file into a text editor and review it. Most lines are commented out by hash marks (#). If you see an option you want to enable, such as an IRQ for a device, uncomment that line. You should also uncomment the line that reads (ACT Y) for the device—this line activates the preceding configuration lines.

> **TIP**
> If your computer dual-boots between Linux and Windows, you may be able to learn of a good configuration by booting Windows and writing down the IRQ, I/O port, and DMA settings used by Windows. Uncomment the equivalent settings in `isapnp.conf` and they'll probably work. If you don't dual-boot to Windows, you may need to locate unused resources by examining `/proc/interrupts`, `/proc/ioports`, and `/proc/dma`. Remember to use every installed hardware device before accessing these files in order to ensure that all your hardware's resources have been activated.

Once you've made these changes, pass the file to the `isapnp` utility, which implements the ISA PnP configuration stored in the configuration file you give it:

```
# isapnp /etc/isapnp.conf
```

Once this is done, if your configuration is correct, you should be able to load the relevant kernel modules and use the device, as described in Chapter 6. If your distribution relies on the `isapnp` package, it should provide a startup script that calls this command so that your hardware is detected automatically at boot time. This package and script is likely to be missing from modern distributions based on the 2.4.*x* or 2.6.*x* kernels, though.

Configuring PCI Devices

The PCI bus was designed with PnP-style configuration in mind; thus, automatic configuration of PCI devices is the rule rather than the exception. For the most part, PCI devices configure themselves automatically and there's no need to make any changes. You can, though, tweak how PCI devices are detected in several ways:

- The Linux kernel has an option that affects how it detects PCI devices: Bus Options ➢ PCI Access Mode. This option has four values: BIOS, which uses the BIOS to do the job; MMConfig, which uses a protocol of that name to detect PCI devices; Direct, which uses a Linux-specific direct-detection system; and Any, which tries MMConfig followed by Direct followed by BIOS detection. In most cases, Any is the most appropriate option; however, if your devices aren't being detected correctly or are being assigned resources that are causing conflicts, you might want to try experimenting with this option.

- Most BIOSes have PCI options that change the way PCI resources are allocated. Adjusting these options may help if you run into strange hardware problems with PCI devices. These options affect only the BIOS detection system, though, and so may work only if you select BIOS as the PCI detection method in the kernel.

- Some Linux drivers support options that cause them to configure the relevant hardware to use particular resources. You should consult the drivers' documentation files for details of

the options they support. You must then pass these options to the kernel using a boot loader (as described later, in "Installing Boot Loaders") or as kernel module options (as described in Chapter 6).

▪ The `setpci` utility can be used to directly query and adjust PCI devices' configurations. This tool is most likely to be useful if you know enough about the hardware to fine-tune its low-level configuration; it's not often used to tweak its basic IRQ, I/O port, or DMA options.

In addition to the configuration options, you may want to check on how PCI devices are currently configured. A command for this purpose is `lspci`, which will display all information about the PCI busses on your system and all devices that are connected to these busses.

Configuring Modems

Modems have long been a staple of computer communications. These devices come in various forms and types, some of which require special hardware configuration. You may need to set your RS-232 serial port hardware to use a conventional telephone modem. Other types of modems support higher-speed communication over telephone or cable TV lines.

Functions of Modems

A modem is a hardware device that enables computers to communicate over ordinary telephone lines. (Some types of modems work over other media, though.) While telephones transmit information in the form of analog waves, computers store and manipulate digital data. A sending modem converts the outgoing digital signal to an analog signal for transmission. This is known as modulation. Likewise, a receiving modem converts the incoming analog signal back to digital for the receiving computer. This is known as demodulation. Hence, the word *modem* is an acronym for modulator/demodulator.

Despite the proliferation of high-speed DSL and cable Internet service, telephone modems are still useful communications devices. Things you can do with your modem include the following:

Log in to a Remote Access Service (RAS) computer or Bulletin Board System (BBS)   Usually, this activity takes the form of a terminal session similar to an old-fashioned dumb terminal and uses a terminal communications program like `minicom`, `kermit` or `cu`.

Connect your computer (or network) to the Internet   This uses a serial implementation of the Internet Protocol (IP), the most common being the *Point-to-Point Protocol (PPP)*. Others include the *Serial Line Internet Protocol (SLIP)* or Compressed SLIP (CSLIP). Chapter 9, "Basic Networking," describes setting up a PPP connection in detail.

Configure your computer for dial-in use   Modems can be used for dial-out purposes, as just described; however, Linux can sit at the other end of this connection. Configuring Linux to accept remote terminal access or incoming PPP connections enables you to give access to your system from remote sites.

Send and receive faxes   Most modern modems have send and receive fax capability.

Setting the RS-232 Serial Port Characteristics

There are times when you have to tell the Linux operating system more information that it can guess from the BIOS about the RS-232 serial port that your modem uses. It may be an internal ISA modem that isn't PnP or it may be simply that the RS-232 serial driver is not detecting things correctly.

> Traditionally, internal modems have been RS-232 serial ports with modem circuitry attached, so you should treat them as ordinary RS-232 serial ports. Today, less complex *software modems* (aka *Winmodems* or *Linmodems*) are more common. These modems don't always work with Linux, as described in the section "Winmodem Detection and Avoidance."

When things are not working correctly, `setserial` is a very useful command for finding out the settings that Linux thinks you have. An example of `setserial` for querying settings looks like this:

```
# setserial /dev/ttyS0
/dev/ttyS0, UART: 16550A, Port: 0x03f8, IRQ: 4
```

This shows the basic information for the RS-232 serial device, including the port and I/O address. More information can be obtained with the `-a` option:

```
# setserial -a /dev/ttyS0
/dev/ttyS0, Line 0, UART: 16550A, Port: 0x03f8, IRQ: 4
        Baud_base: 115200, close_delay: 50, divisor: 0
        closing_wait: 3000
        Flags: spd_normal skip_test
```

You can also view the settings of multiple RS-232 serial ports. In a sample system with two RS-232 serial ports and a third one due to an internal modem, you would use a command like this:

```
# setserial -a /dev/ttyS[014]
/dev/ttyS0, UART: 16550A, Port: 0x03f8, IRQ: 4
/dev/ttyS1, UART: 16550A, Port: 0x02f8, IRQ: 3
/dev/ttyS4, UART: 16550A, Port: 0xdc00, IRQ: 5
```

You may notice that the `/dev/ttyS4` device is using a non-standard I/O address and IRQ (it's an internal modem). If you need to change a setting, you can do so by passing it to `setserial`. For example, if the real IRQ for `/dev/ttyS4` was 10, you would use a command like this:

```
# setserial /dev/ttyS4 IRQ 10
```

For adjusting settings, `setserial` takes a large number of options; consult its `man` page for details (that is, type **`man setserial`**). You can adjust the IRQ, the connection speed, and more.

> **NOTE** Although `setserial` provides options to set IRQs and other low-level features, these options don't set features in the sense that the PnP tools do. Instead, `setserial` tells the system how the card is already configured in case it's been detected incorrectly.

If your communication speed is set incorrectly, you can adjust it with a `setserial` command:

```
# setserial /dev/ttyS0 baud_base 9600
```

This command sets the RS-232 serial port's connect speed to 9,600 bits per second (bps). The term *baud* is often used as synonymous with bps. Technically there are differences, but `setserial` uses baud in various option names when bps would be more appropriate.

> **NOTE** Most programs that use modems provide their own means to set the modem's speed; thus, you're unlikely to need to use `setserial` to change the modem's speed in most configurations. A few very simple tools do require this use, though.

Frequently, `/dev/modem` is a link to one of the `/dev/ttySx` files. This practice is intended to simplify access to the modem; you can refer to `/dev/modem` rather than the actual device file when using `setserial` or other modem-using commands. Do not confuse a real RS-232 serial device file with `/dev/modem`; although they often work alike, `/dev/modem` is not guaranteed to point to the real modem device file.

Part of the output to the `setserial` command is a field called `uart`. A Universal Asynchronous Receiver/Transmitter (UART) is a controller chip that processes data coming in from and going out of an RS-232 serial device. Any UART versions prior to the 16550 and 16550A lack a buffer that lets you get higher communication speeds with your modems. Typical slower UARTs are the 8250 and 16450, and it is best to keep them at a speed of 9,600bps or lower. All modern computers' RS-232 serial ports have at least 16550A-compatible UARTs, but you might run into a less-capable UART on an old modem or RS-232 serial port board you've found in a parts bin.

Winmodem Detection and Avoidance

Most modern internal modems are not full hardware modems—that is, they're devices in which some of the modem functionality is not built into the card. These modems rely on software drivers that offload some of the modem duty to the main microprocessor of the computer system. The modems are generally called Winmodems or software modems (although the term Linmodems is sometimes applied when they can be made to work in Linux), and they require additional software to operate. Check `http://www.linmodems.org` for details.

Similarly, most USB-interfaced modems use custom chipsets and require custom drivers that aren't available for Linux. One major exception is USB modems that use the Communication Device Class/Abstract Control Modem (CDC ACM) interface; these modems work in Linux.

In the case of both software modems and proprietary USB modems, the trick is identifying them. Examining a box in a store, you may be hard-pressed to tell a software modem from a hardware modem. One clue is the "requirements" list on the box. If the box specifies that the modem works with DOS, OS/2, Linux, or other non-Windows OSs, chances are it's a hardware modem. If the box lists only Windows as a supported OS, chances are it's a software modem. (On the other hand, the manufacturer might just be reluctant to deal with non-Windows support and so might list only Windows as a supported OS.)

If the modem is already installed in a computer, you may just need to boot the system to check it out. If you can get the modem to respond using a standard /dev/ttySx device file, chances are it's a hardware modem. Information obtained from pnpdump or lspci might also provide clues—but just because the device is detected by these tools does not mean that it's a hardware modem.

You may also want to consult online resources. The Linmodems website contains information on many supported and unsupported software modems, so it's a good starting point. Your distribution's hardware compatibility list (usually available from your distribution's website) may be helpful as well.

Using Broadband Hardware

The word *modem* is most frequently applied to analog telephone modems; however, other devices serve similar functions but provide faster speed. These devices, or the services they provide, are often referred to as *broadband*. This term refers to either high-speed remote network access or a communications medium that can deliver multiple types of data (such as voice, video, and digital data). The most common types of broadband hardware are as follows:

ISDN adapters   The *Integrated Services Digital Network (ISDN)* protocol is a digital telephone variant. In its basic form, it provides for two 64Kbps data channels plus an extra 16Kbps channel for control signals, for a total of 144Kbps in both directions (although only 128Kbps is usable for data)—over twice the 56Kbps a typical telephone modem provides. An ISDN adapter works much like a conventional telephone modem but connects to an ISDN line. Some ISDN adapters connect to an RS-232 serial port, but others are internal devices that require special Linux drivers. (These are available in the Device Drivers ➢ ISDN Subsystem area of the kernel configuration.)

DSL modems   *Digital Subscriber Line (DSL)* is a term that's applied to a wide range of connection protocols delivered over conventional telephone lines. Depending on the variety, DSL can provide speeds of several megabits per second (Mbps). Because of this very high speed, DSL modems don't usually connect through an RS-232 serial port; they use an Ethernet connection, a USB port, or an internal (PCI) expansion slot. Ethernet-connected DSL modems can be treated just like ordinary network connections, but USB-interfaced and internal DSL modems require special drivers.

Cable modems   Cable modems are just like DSL modems from Linux's point of view, although the underlying technology is quite different. Most cable modems are external Ethernet-interfaced devices, although some use USB ports and a few internal cable modems exist. As with DSL modems, the external Ethernet-based devices are the simplest to configure.

A decade ago, ISDN held great promise, and it made some headway in Europe. ISDN never caught on in the U.S., though, and with the rise of faster forms of broadband, ISDN seems doomed to obscurity. DSL and cable modems are both quite popular, though. If you're shopping for one of these services, be sure you get an external Ethernet-interfaced modem. This will enable you to use a conventional Ethernet card, for which Linux drivers are plentiful. If your provider insists on giving you an internal or USB-interfaced device, you'll need to go hunting for drivers, and there's no guarantee that you'll find them.

On the software end, ISDN modems work just like telephone modems; you run the standard Linux PPP utilities, as described in Chapter 9, to make a connection to an Internet service provider (ISP). If you're using an internal ISDN adapter, though, you'll need to use the appropriate device identifier rather than a `/dev/ttySx` RS-232 serial port identifier.

Some DSL and most cable modem connections work just like local network connections; you configure your system to use a static IP address or to obtain an IP address automatically, as described in Chapter 9. If your modem is an Ethernet-interfaced model, there's nothing else to it. If your modem is an internal or USB-interfaced device, though, you'll need to consult your driver's documentation to learn what network device name you'll use.

Many DSL connections don't use standard local network protocols. Instead, they use *PPP over Ethernet (PPPoE)*, which is a variant of PPP designed for use over Ethernet rather than over RS-232 serial lines. A similar protocol, PPP over ATM (PPPoA), is also used with some internal modems. Most modern Linux distributions ship with PPPoE software, and recent kernels include this support directly. You can also check for add-on packages, such as Roaring Penguin's RP-PPPoE (`http://www.roaringpenguin.com/pppoe/`).

Configuring USB Devices

Modern computers invariably ship with *Universal Serial Bus (USB)* ports, and increasing numbers of external devices are now being produced in USB form, often replacing older interfaces as the preferred method of connecting to a computer. This fact means you must understand something about USB, including USB itself, Linux's USB drivers, and Linux's USB management tools.

USB Basics

USB is a protocol and hardware port for transferring data to and from devices. It allows for many more (and varied) devices per interface port than either ATA or SCSI and gives better speed than RS-232 serial and parallel ports. The USB 1.0 and 1.1 specifications allow for up to 127 devices and 12Mbps of data transfer. USB 2.0 allows for much higher transfer rates—480Mbps, to be precise.

> **NOTE** Data transfer speeds may be expressed in bits per second (bps) or multiples thereof, such as megabits per second (Mbps), or in bytes per second (B/s) or multiples thereof, such as megabytes per second (MB/s). In most cases, there are 8 bits per byte, so multiplying or dividing by eight may be necessary if you're trying to compare speeds of devices that use different measures.

USB is the preferred interface method for many external devices, including printers, scanners, mice, digital cameras, and music players. USB keyboards, Ethernet adapters, modems, speakers, hard drives, and other devices are also available, although USB has yet to dominate these areas as it has some others.

Most computers ship with four to eight USB ports. (A few years ago, two USB ports were more common.) Each port can handle one device by itself, but you can use a *USB hub* to connect several devices to each port. Thus, you can theoretically connect huge numbers of USB devices to a computer. In practice, you may run into speed problems, particularly if you're using USB 1.*x* for devices that tend to transfer a lot of data, such as scanners, printers, or hard drives.

> If you've got an older computer that only supports USB 1.*x* and you want to connect a high-speed USB 2.0 device, buy a separate USB 2.0 board. You can use this board for the USB 2.0 device and either disable the onboard USB 1.*x* ports or use them for slower devices.

Linux USB Drivers

Three competing controller types for USB are common:

UHCI   The *Universal Host Controller Interface (UHCI)* is a USB 1.*x* controller most commonly found on motherboards with Intel or VIA chipsets.

OHCI   The *Open Host Controller Interface (OHCI)* is a USB 1.*x* controller most commonly found on non-Intel and non-VIA motherboard chipsets and in many add-on cards.

EHCI   The *Enhanced Host Controller Interface (EHCI)* is the most common type of USB 2.0 controller.

Motherboards and USB add-on cards that support USB 2.0 almost always provide both EHCI and either UHCI or OHCI hardware. Thus, you must ensure that your kernel supports both EHCI and either UHCI or OHCI; if you load just the EHCI driver, you won't be able to use USB 1.*x* hardware, and if you omit the EHCI driver, you'll be able to use USB 2.0 devices only at the much lower USB 1.*x* speeds. Supporting both UHCI and OHCI won't cause problems; the kernel will simply detect which one your system has and use the correct driver. In the past, USB driver modules were called `usb-uhci.o` and `usb-ohci.o` for UHCI and OHCI, respectively; however, with more modern kernels they're called `uhci-hcd.ko`, `ohci-hcd.ko`, and `ehci-hcd.ko`.

You can learn a great deal about your devices by examining the `/proc/bus/usb/devices` file—type **less /proc/bus/usb/devices** to view it in the `less` pager, for instance. Much of the information in this file will look like gibberish unless you're familiar with the details of USB implementations, but even the uninitiated can extract some useful clues. Pay particular attention to the `Manufacturer`, `Product`, and `Driver` items, which contain information on the manufacturer, model, and driver used for the device, respectively. Note that several of the entries in this file are for the USB controller itself. Others may belong to any USB hubs you have.

Early Linux USB implementations required a separate driver for every USB device. In fact, many of these drivers remain in the kernel, and some software relies on them. For instance, USB disk storage devices use USB storage drivers that interface with Linux's SCSI support, making USB hard disks, removable disks, and so on look like SCSI devices. Linux is migrating, though, toward a model in which a USB filesystem provides access to USB devices. This filesystem appears as part of the `/proc` virtual filesystem. In particular, USB device information is accessible from `/proc/bus/usb`. That's why `/proc/bus/usb/devices` holds basic device information. Subdirectories of `/proc/bus/usb` are given numbered names based on the USB controllers installed on the computer, as in `/proc/bus/usb/001` for the first USB controller. Software can access files in these directories to control USB devices rather than use device files in `/dev` as with most hardware devices.

If you want to know what driver is associated with a specific USB device's `/proc/bus/usb` entry, you can use the `usbmodules` program and its `--device` option:

```
$ usbmodules --device /proc/bus/usb/005/004
usblp
usb-storage
```

This example identifies the device as using the `usblp` and `usb-storage` modules, which are used for printers and storage (disk) devices, respectively. This specific device is a multi-function printer/scanner with an integrated media reader, so both printer and storage drivers apply to it. Not all devices have drivers that are identified in this way; just devices with kernel-level drivers for the devices are so identified. Devices that are accessed directly by their respective utility programs have no drivers. Sometimes, a device can be accessed both through a kernel-level driver and through direct means. For instance, the preceding example shows no driver for the scanner functions of the device; those are handled by direct access.

USB Manager Applications

USB can be challenging for OSs because it was designed as a *hot-pluggable* technology—that is, you can connect or disconnect USB devices while the computer is powered up and working. The Linux kernel was not originally designed with this sort of activity in mind, so the kernel relies on external utilities to help manage matters. Two tools in particular are used for managing USB devices: `usbmgr` and `hotplug`.

The `usbmgr` package (located at `http://www.dotaster.com/~shuu/linux/usbmgr/`) is a program that runs in the background to detect changes on the USB bus. When it detects changes, it loads or unloads the kernel modules that are required to handle the devices. For instance, if you plug in a USB Zip drive, `usbmgr` will load the necessary USB and SCSI disk modules. This package uses configuration files in `/etc/usbmgr` to handle specific devices and `/etc/usbmgr/usbmgr.conf` to control the overall configuration.

With the shift from in-kernel device-specific USB drivers to the USB device filesystem (`/proc/bus/usb`), `usbmgr` has been declining in importance. In fact, it may not be installed at all on your system. Instead, most distributions rely on the Hotplug package (`http://linux-hotplug.sourceforge.net`), which relies on kernel support added with the 2.4.*x* kernel series. This

system uses files stored in /etc/hotplug to control the configuration of specific USB devices. In particular, /etc/hotplug/usb.usermap contains a database of USB device IDs and pointers to scripts in /etc/hotplug/usb that are run when devices are plugged in or unplugged. These scripts might change permissions on USB device files so that ordinary users can access USB hardware, run commands to detect new USB disk devices, or otherwise prepare the system for a new (or newly removed) USB device.

Configuring Sound Cards

There are two main sound systems for Linux: the *Open Sound System (OSS)* and the *Advanced Linux Sound Architecture (ALSA)*. Prior to the 2.6.*x* kernel series, OSS was the only sound system built into the standard Linux kernel; ALSA was an optional add-on available from its website, http://www.alsa-project.org. ALSA was added to the main Linux kernel with the 2.6.*x* kernel series, and OSS is now deprecated. You might find that OSS works better for you, but in most cases ALSA is at least as good. If you have a choice, you should probably use ALSA.

> A commercial variant of OSS is available from 4Front Technologies (http://www.opensound.com). This package is fully compatible with the kernel's standard sound system, but some of the drivers work better than the standard kernel OSS drivers. It's worth investigating if neither the kernel's OSS nor the ALSA drivers work for your sound card.

At the hardware level, sound cards are just like any other devices and can be configured through tools such as isapnp or the hardware drivers. You may need to consult the documentation for your particular drivers to learn about driver-specific kernel or module options.

Beyond the low-level hardware options, several command-line and GUI tools exist to help your system detect the sound hardware and load the correct modules:

sndconfig   This program was developed by Red Hat as a way to probe for a sound card and set up appropriate configuration files. It was adopted by several other distributions, but has since been supplanted by other tools. If your distribution still supports it, type **sndconfig** as root to run the program. You can limit its actions by including the --noprobe option (which prevents the system from probing for PnP cards) or --noautoconfig (which prevents the system from configuring PnP cards). If you use these options, you'll have to enter configuration parameters manually.

Audio Devices   This program, also known as system-config-soundcard or redhat-config-soundcard, is the successor to sndconfig on Fedora and Red Hat systems. To use it, type its name or locate its option in the default desktop menu. The result should be a dialog box similar to the one shown in Figure 3.4. You can test the detected sound hardware by clicking Play Test Sound. Unfortunately, this tool, like most such tools, presents little in the way of debugging or configuration options.

F I G U R E 3 . 4   When they work, GUI audio configuration tools provide simplified detection and configuration of sound cards.
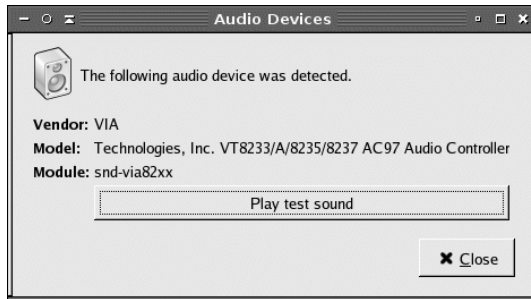
YaST and YaST2   SuSE's YaST and YaST2 tools provide a wide range of configuration options using text-mode and GUI interfaces, respectively. In particular, the Sound option in the Hardware area does the job; it should detect the sound card and change the necessary configuration files to have it work every time the system boots.

alsactl   This program is less of a detection tool than it is a tool to set the sound hardware's parameters. It's an ALSA-only tool that can read or set the values for the sound card's mixer (that is, its volume settings). Type **alsactl store** to store the current mixer settings in its configuration file (/etc/asound.state); type **alsactl restore** to change the mixer settings to the values stored in this file. Distributions with good ALSA support generally use this command to set up ALSA's mixer when the system boots, so setting the volume as you like it and then typing **alsactl store** will restore the settings automatically the next time you boot.

Configuring sound hardware is usually either extremely easy or extremely hard. In most cases, tools like sndconfig, Audio Devices, or YaST will correctly detect your hardware and get it working. Once this is done, the hardware itself is configured, although of course you must use audio players, mixers, and other tools to play sounds, adjust the output, and so on.

> You can use the play program to test basic sound-playing capabilities. This program plays most audio files with .wav filename extensions.

Unfortunately, sometimes the automated audio-detection and auto-configuration tools fail. When this happens, you must figure out which driver to load and load it in some other way, such as compiling it into your kernel or manually loading the appropriate driver module. (These topics are covered in Chapter 6.)

> ⊕ **Real World Scenario**
>
> **Getting ALSA to Work**
>
> If you elect to use ALSA drivers, you may run into a problem: You'll load the drivers and test a sound file with `play` or its ALSA-specific variant, `aplay`. The program won't return an error message, but neither will it produce any sound.
>
> The trick is that ALSA starts up with all audio channels muted by default. To fix the problem, you must use a mixer program, such as `alsamixer`, to unmute the relevant channels. (You may need to unmute several channels, such as a master channel and a PCM device.) Once this is done, you should be able to hear sounds.
>
> To make your changes permanent, type **`alsactl store`**. If sound is still muted when you reboot the computer, you'll need to modify a startup script to include a call to `alsactl restore`.

Configuring SCSI Devices

The *Small Computer System Interface (SCSI)* is an interface standard for connecting peripheral devices to your computer system. Typical devices include hard disks, tape drives, and CD-ROM drives. SCSI competes with the ATA standard. In most respects, SCSI is technically superior to ATA, but ATA is less expensive and is better supported on *x*86 hardware. Thus, SCSI is often used on servers and other high-performance systems, whereas ATA is most often used on desktop systems. Like ATA, SCSI hardware comes in several varieties, so you should understand something about the different SCSI varieties. Actually configuring SCSI devices requires knowing something about their addressing systems and termination. Once a system with SCSI devices is booted, you may need to identify specific devices, and of course Linux provides tools to help you do this.

Varieties of SCSI

There are many types of SCSI definitions from the commonly know SCSI-1, SCSI-2, and SCSI-3 (also known as Ultra Wide SCSI) to the often confused Fast SCSI, Wide SCSI, Fast Wide SCSI, UltraSCSI, and more. One critically important difference between SCSI busses is the size of the SCSI bus, which comes in 8-bit and 16-bit widths. The 8-bit SCSI variants are sometimes referred to as Narrow SCSI, while the 16-bit versions are sometimes called Wide SCSI. A Wide SCSI bus supports twice as much data being transferred at a time, so these varieties are faster than their 8-bit counterparts at any given bus speed. Wide SCSI also requires cables with more wires (80-pin versus 40- or 25-pin for Narrow SCSI). Despite the additional wires, Wide SCSI cables are often physically smaller than Narrow SCSI cables, because the connectors pack the pins in more densely.

> Traditional varieties of both ATA and SCSI use parallel busses, meaning that several signal lines run in parallel, enabling the transfer of an entire byte of data (or more) with each exchange. Parallel busses may sound more efficient than the alternative serial busses, in which a single bit is transferred at a time; but in practice, timing problems in the design of parallel busses can slow things down. Thus, both ATA and SCSI are moving toward serial designs, with *Serial ATA (SATA)* and *Serial Attached SCSI (SAS)*. In the future, these two busses seem likely to merge, resulting in a single standard for both ATA and SCSI.

An 8-bit SCSI bus has three address lines, which allows up to 8 devices to be on that SCSI bus (2^3 = 8 combinations). A 16-bit SCSI bus has 4 address lines and allows for up to 16 devices on that SCSI bus (2^4 = 16 combinations). In both cases, the SCSI host adapter itself counts as 1 device, so excluding the SCSI adapter, Narrow SCSI permits up to 7 devices while Wide SCSI supports up to 15 devices—in theory. In practice, cable length limits make it hard to fit more than 3 to 5 devices on most varieties of SCSI. (Cable length limits vary from one variety of SCSI to another. Consult your host adapter's documentation for details.)

SCSI was designed with backward compatibility in mind. In theory, you can attach any type of SCSI device to any type of SCSI host adapter. (Low-voltage differential, or LVD, SCSI is an exception to this rule, as is the new SAS.) In practice, you're best off sticking to devices that are similar in protocol levels; mixing something very old (such as an old SCSI-1 CD-ROM drive) with something new (such as the latest Ultra640 SCSI hard disk) is likely to cause problems.

> SCSI hard disks aren't detected by the standard *x*86 BIOS. You can still boot from a SCSI hard disk *if* your SCSI host adapter has its own BIOS that supports booting. Most high-end SCSI host adapters have this support, but low-end SCSI host adapters don't have BIOSes. If you use such a host adapter, you can still attach SCSI hard disks to the adapter, and Linux can use them, but you'll need to boot from an ATA hard disk.

SCSI IDs

In order for the host adapter to identify and communicate with SCSI devices, each device, including the host adapter, is given an ID number. For 8-bit SCSI these IDs are in the range 0 through to 7. For 16-bit SCSI they are numbered 0 through 15.

When SCSI was only available as an 8-bit bus, SCSI IDs were used to also indicate priority order for attention on the bus. For this reason, SCSI ID 7 is the best choice for the SCSI host adapter itself and all lower SCSI IDs have an increasingly lower priority. When 16-bit SCSI devices were developed, the original 7 through to 0 priorities were kept as the high-priority IDs and the remaining IDs from 15 through to 8 are left as the lower-priority IDs.

> **NOTE** SCSI priorities vary from one host adapter to another. Although the system just described applies to most host adapters, some deviate from this system.

SCSI IDs are not used to identify the corresponding device file on a Linux system. Hard drives follow the naming of /dev/sdx (where x is a letter from a up), SCSI tapes are given /dev/stx and /dev/nstx (where x is a number from 0 up), and SCSI CD-ROMs are /dev/scdx (where x is a number from 0 and up).

SCSI device identification can be confusing at first, but the important thing to remember is that the Linux device numbering (or lettering) is assigned in increasing order based on the SCSI ID. If you had one hard disk with a SCSI ID of 2 and another hard disk with a SCSI ID of 4, they will be assigned to /dev/sda and /dev/sdb, respectively. The real danger here is if you happen to add a third SCSI drive and give it an ID of 0, 1, or 3. This new disk would become /dev/sda (for IDs of 0 or 1) or /dev/sdb (for ID 3), bumping up one or both of the existing disks' Linux device identifiers. For this reason, it's usually best to give hard disks the lowest possible SCSI IDs so that you can add future disks using higher IDs.

Another complication is when you have multiple SCSI host adapters. In this case, Linux assigns device filenames to all of the disks on the first adapter followed by all those on the second adapter. Depending on where the drivers for the SCSI host adapters are found (compiled directly into the kernel or loaded as modules) and how they're loaded (for modular drivers), you might not be able to control which adapter takes precedence.

> **WARNING** Remember that some non-SCSI devices, such as USB disk devices, are mapped onto the Linux SCSI subsystem. This can cause a true SCSI hard disk to be assigned a higher value than you'd expect if you use USB devices. The same is true for some varieties of SATA drives, because many Linux SATA drivers are treated like SCSI drivers by the Linux kernel.

SCSI Termination

The SCSI bus is logically one-dimensional—that is, every device on the bus falls along a single line. This bus must not fork or branch in any way. Figure 3.5 shows both good and bad SCSI bus configurations.

Each end of the SCSI bus must be *terminated*. This refers to the presence of a special resistor pack that prevents signals from bouncing back and forth along the SCSI chain. Several types of SCSI termination have been developed over the years, including simple passive termination, active termination, and low-voltage differential (LVD) termination. Consult your SCSI host adapter and SCSI devices' manuals to learn what type of termination they require. Remember that both ends of the SCSI chain must be terminated, but devices mid-chain must *not* be terminated. The SCSI host adapter qualifies as a device, so if it's at the end of the chain, it must be terminated.

FIGURE 3.5 A SCSI bus should be a single line, with no branches or forks.
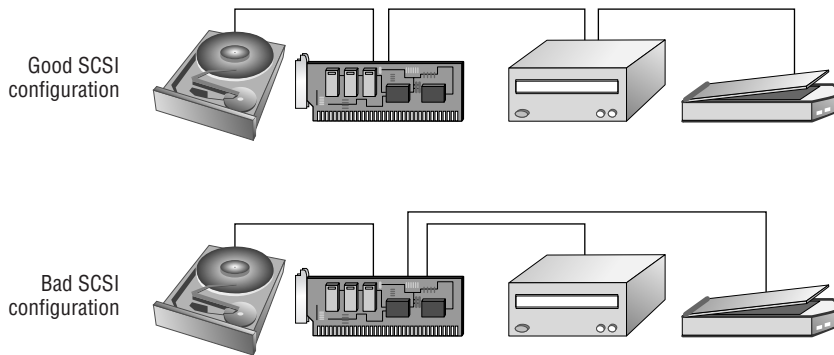
Modern SCSI devices usually include a jumper or other setting to enable or disable termination. Older internal devices use comb-shaped resister packs that you insert or remove to terminate the device, and older external devices use external terminators that attach to one of the device's two SCSI connectors. (A few external SCSI devices are permanently terminated and have just one SCSI connector.)

Incorrect termination often results in bizarre SCSI problems, such as an inability to detect SCSI devices, poor performance, or unreliable operation. Similar symptoms can result from the use of poor-quality SCSI cables or cables that are too long.

Identifying SCSI Hardware

Information on your SCSI devices is kept in the /proc/scsi/scsi file. Inspecting it will give you a summary of the manufacturers, model numbers, SCSI IDs, and similar information for all your SCSI devices:

```
$ cat /proc/scsi/scsi
Attached devices:
Host: scsi0 Channel: 00 Id: 05 Lun: 00
  Vendor: IOMEGA   Model: ZIP 100           Rev: D.13
  Type:   Direct-Access                     ANSI SCSI revision: 02
Host: scsi2 Channel: 00 Id: 00 Lun: 00
  Vendor: ATA      Model: Maxtor 6Y080M0    Rev: 1.02
  Type:   Direct-Access                     ANSI SCSI revision: 05
```

This example demonstrates the presence of two SCSI devices. If you examine it closely, you'll notice something odd: The second SCSI device is identified with a vendor code of ATA. In truth, this is an SATA hard disk, not a SCSI disk. The driver for this device just happens to tie itself into the SCSI subsystem, making the disk look just like a SCSI hard disk to Linux. The first disk is a genuine SCSI disk, though. Because of this mix of devices, Linux thinks it has two SCSI host adapters installed (scsi0 and scsi2).

To further complicate matters, each SCSI device can contain multiple logical unit numbers (LUNs). These are typically used by sophisticated devices such as RAID controllers, tape drives, and CD-ROM units that support multiple media or other devices. In these cases, the various LUNs will be reported along with the SCSI ID to the controller so that these devices may be accessed individually.

Designing a Hard Disk Layout

Whether your system uses ATA or SCSI disks, you must design a disk layout for Linux. If you're using a system with Linux preinstalled, you may not need to deal with this task immediately; however, sooner or later you'll have to install Linux on a new computer or one with an existing OS or upgrade your hard disk. The next few pages describe the *x*86 partitioning scheme, Linux mount points, and common choices for a Linux partitioning scheme. The upcoming section "Creating Partitions and Filesystems" covers the mechanics of creating partitions.

Why Partition?

The first issue with partitioning is the question of why it should be done. The answer is that partitioning provides a variety of advantages, including:

Multi-OS support   Partitioning enables you to keep the data for different OSs separate from each other. In fact, many OSs can't easily co-exist on the same partition because they don't support each other's primary filesystems. This feature is obviously important only if you want the computer to boot multiple OSs. It can still be handy to help maintain an emergency system, though—you can install a single OS twice, using one installation as an emergency maintenance tool for the first in case problems develop.

Filesystem choice   By partitioning your disk, you can use different *filesystems*—data structures designed to hold all the files on a partition—on each partition. Perhaps one filesystem is faster than another and so is important for time-critical or frequently accessed files, but another might provide accounting or backup features you want to use for users' data files.

Disk space management   By partitioning your disk, you can lock certain sets of files into a fixed space. For instance, if you restrict users to storing files on one or two partitions, they can fill those partitions without causing problems on other partitions, such as system partitions. This feature can help keep your system from crashing if space runs out. On the other hand, if you get the partition sizes wrong, it can cause lesser problems much sooner than would be the case if you'd used fewer or no partitions.

Disk error protection   Disks sometimes develop problems. These problems can be the result of bad hardware or of errors that creep into the filesystems themselves. In either case, splitting a disk into partitions provides some protection against such problems. If data structures on one partition become corrupted, these errors affect only the files on that disk. This separation can therefore protect data on other partitions and simplify data recovery.

Security   You can use different security-related mount options on different partitions. For instance, you might mount a partition that holds critical system files read-only, preventing users from writing to that partition. Linux's file security options should provide similar protection, but taking advantage of Linux filesystem mount options provides redundancy that can be helpful in case of an error in setting up file or directory permissions.

Backup   Some backup tools work best on whole partitions. By keeping partitions small, you may be able to back up more easily than you could if your partitions were large.

In practice, most Linux computers use several partitions, although precisely how the system is partitioned varies from one system to another. (The upcoming section "Common Partitions and Filesystem Layouts" describes some common possibilities.)

Types of Disk Partitions

The x86 architecture dates back to the 1980s, when a big hard disk was about 10MB in size. In such an environment, creating lots of partitions on the disk seemed pointless, so the original x86 partitioning scheme supported just four partitions. Today, these four partitions are known as *primary partitions*. Some OSs, such as DOS and Windows, must boot from a primary partition. Linux isn't so limited, but using one or more primary partitions for some (or even all) of Linux's partition needs is common.

As hard disks grew larger and OSs more plentiful, the four-partition limit of the original x86 partitioning scheme became a problem. To work around the issue while maintaining backward compatibility, the x86 partitioning scheme was expanded by using a single primary partition as a placeholder for an arbitrary number of additional partitions. The placeholder partition is known as an *extended partition,* and the partitions it contains are called *logical partitions.* The x86 partitioning system supports an arbitrary number of logical partitions, but because they're all contained within a single extended partition, the logical partitions must all be contiguous with one another. Figure 3.6 depicts this scheme, although the specific partition sizes, locations, and number of partitions are arbitrary.

Under Linux, primary and extended partitions are assigned numbers from 1 to 4, as in `/dev/hda1` or `/dev/sdb3` for the first primary partition on the first ATA disk or the third primary partition on the second SCSI disk, respectively. These numbers are fixed, and numbers can be skipped. For instance, a disk might have `/dev/hda1` and `/dev/hdb3`, but not `/dev/hda2` or `/dev/hda4`.

FIGURE 3.6   The x86 partitioning system uses up to four primary partitions, one of which can be a placeholder extended partition that contains logical partitions.
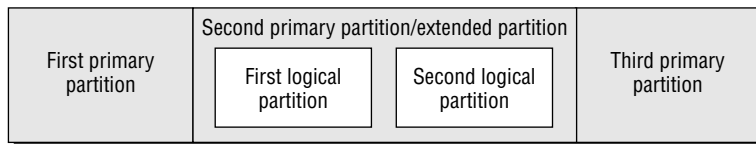
Logical partitions are assigned numbers from 5 up, as in /dev/hda5 or /dev/sdb5. These numbers are assigned sequentially, and numbers aren't ordinarily skipped; if a disk has /dev/hda6, a /dev/hda5 also exists.

For the most part, Linux doesn't care whether a partition is primary or logical; you can use either partition for just about any partition function in Linux. For booting the OS, you can use either type of partition, but if you put certain Linux files on a logical partition, you'll limit your boot loader options. (This topic is described in more detail later in this chapter, in "Installing Boot Loaders.")

In addition to the distinction between primary, extended, and logical partitions, another class of partition types is important: The *x*86 partition table supports partition type codes, which are two-digit hexadecimal numbers that are assigned to specific functions. For instance, 0x06 is reserved for a certain type of File Allocation Table (FAT) partition, 0x82 denotes a Linux swap partition, and 0x83 indicates a Linux filesystem partition. Some OSs, such as DOS and Windows, rely on these type codes to determine which partitions they should try to access. DOS and Windows ignore Linux partitions because of their type codes, for instance. For the most part, though, Linux ignores partition type codes; you can try to mount any type of partition. A couple of partial exceptions to this rule exist, though:

- During installation, most Linux distributions pay attention to the partition type codes to help them guess how the system is configured. Installers and Linux disk utilities also create partitions with appropriate type codes set.

- Linux relies on the extended partition type codes (0x05 and 0x0f) to identify extended partitions. You can still try to act directly on the extended partition using various utilities, but for the most part doing so would be a mistake.

The *x*86 partition table stores partition information in both CHS and LBA forms. The partition table also records the CHS geometry for the disk. If you use a disk on just one computer, no problems will result; however, if you move a disk from one computer to another, the CHS geometry might change, which can result in confusion. Linux is usually pretty flexible about this, so Linux can usually cope well with such changes. Other OSs aren't always so flexible, though.

Although partitioning is usually considered a hard disk tool, some other disk devices can be partitioned. This practice is most common with certain types of removable disks. Because most removable disks are treated just like hard disks in Linux, you can partition them as you see fit, and in fact this practice is common with certain types of removable disks, such as Zip disks. Other removable disks, such as CD-ROMs and magneto-optical (MO) disks, are not commonly partitioned. Linux doesn't support partitioning for some types of disks, such as floppy disks. If you're in doubt about whether to partition a particular type of removable disk, try examining a working disk. Type **fdisk -l /dev/hdb** (change hdb to an appropriate device identifier) to examine the partition table on a disk. If this command returns partition table information, other disks of this type should probably be partitioned. The upcoming section "Partitioning a Disk" describes how to use fdisk in more detail.

Most non-*x*86 platforms have their own partitioning systems. Most of these systems are simpler than the *x*86 system in that there's no distinction between primary, extended, and logical partitions. The basic principles of partitioning are the same across these platforms, though: The disk is split into multiple partitions for the same reasons *x*86 systems are partitioned, and Linux identifies the partitions in the same way (as /dev/hda2, for instance). Thus, at a system level, non-*x*86 partitions work just like *x*86 partitions.

To manipulate non-*x*86 partitions, though, you may use different tools. Sometimes these tools have the same name as the *x*86 tools, but their operational details may differ. One tool that's common across several platforms is GNU Parted (http://www.gnu.org/software/parted/), which supports *x*86, Macintosh, and several other types of partition tables.

Linux itself can handle about a dozen different partitioning schemes, although *x*86 distributions may not have the necessary support compiled by default. The File Systems ➢ Partition Types kernel configuration area lists the available options should you need to read disks created on an unusual platform.

Mount Points

Once a disk is partitioned, an OS must have some way to access the data on the partitions. In DOS and Windows, this is done by assigning a drive letter, such as C: or D:, to each partition. (DOS and Windows use partition type codes to decide which partitions get drive letters and which to ignore.) Linux, though, doesn't use drive letters; instead, Linux uses a unified directory tree. Each partition is *mounted* at a *mount point* in that tree. A mount point is simply a directory that's used as a way to access the filesystem on the partition, and mounting the filesystem is the process of linking the filesystem to the mount point.

For instance, suppose that a Linux system has three partitions: the root (/) partition, /home, and /usr. The root partition holds the basic system files, and all other partitions are accessed via directories on that filesystem. If /home contains users' home directories, such as sally and sam, those directories will be accessible as /home/sally and /home/sam once this partition is mounted at /home. If this partition were unmounted and remounted at /users, the same directories would become accessible as /users/sally and /users/sam.

Partitions can be mounted just about anywhere in the Linux directory tree, including on directories on the root partition as well as directories on mounted partitions. For instance, if /home is a separate partition, you could have a /home/morehomes directory that serves as a mount point for another partition.

Chapter 4, "Managing Files and Filesystems," describes the commands and configuration files that are used for mounting partitions. For now, you should only be concerned with what constitutes a good filesystem layout (that is, what directories you should split off into their own partitions) and how to create these partitions.

Common Partitions and Filesystem Layouts

So, what directories are commonly split off into separate partitions? Table 3.3 summarizes some popular choices. Note that typical sizes for many of these partitions vary greatly depending on how the system is used. Therefore, it's impossible to make recommendations on partition size that will be universally acceptable.

TABLE 3.3 Common Partitions and Their Uses

Partition (mount point)	Typical size	Use
Swap (not mounted)	1.5–2 times system RAM size	Serves as an adjunct to system RAM; is slow, but enables the computer to run more or larger programs.
/home	200MB–200GB	Holds users' data files. Isolating it on a separate partition preserves user data during a system upgrade. Size depends on number of users and their data storage needs.
/boot	5–50MB	Holds critical boot files. Creating as a separate partition allows for circumventing limitations of older BIOSes and boot loaders on hard disks over 8GB.
/usr	500MB–6GB	Holds most Linux program and data files; this is frequently the largest partition.
/usr/local	100MB–3GB	Holds Linux program and data files that are unique to this installation, particularly those that you compile yourself.
/opt	100MB–3GB	Holds Linux program and data files that are associated with third-party packages, especially commercial ones.
/var	100MB–200GB	Holds miscellaneous files associated with the day-to-day functioning of a computer. These files are often transient in nature. Most often split off as a separate partition when the system functions as a server that uses the /var directory for server-related files like mail queues.
/tmp	100MB–20GB	Holds temporary files created by ordinary users.
/mnt	N/A	/mnt isn't itself a separate partition; rather, it or its subdirectories are used as mount points for removable media like floppies or CD-ROMs.
/media	N/A	Holds subdirectories that may be used as mount points for removable media, much like /mnt or its subdirectories.

Some directories—/etc, /bin, /sbin, /lib, and /dev—should *never* be placed on separate partitions. These directories host critical system configuration files or files without which a Linux system cannot function. For instance, /etc contains /etc/fstab, the file that specifies what partitions correspond to what directories, and /bin contains the mount utility that's used to mount partitions on directories.

> **NOTE** The 2.4.*x* and later kernels include support for a dedicated /dev filesystem, which obviates the need for files in an actual /dev directory, so in some sense, /dev can reside on a separate filesystem, although not a separate partition.

Creating Partitions and Filesystems

If you're installing Linux on a computer, chances are it will present you with a tool to help guide you through the partitioning process. These installation tools will create the partitions you tell them to create or create partitions sized as the distribution's maintainers believe appropriate. If you need to partition a new disk you're adding, though, or if you want to create partitions using standard Linux tools rather than rely on your distribution's installation tools, you must know something about the Linux programs that accomplish this task. Partitioning really involves two tasks: creating the partitions and preparing the partitions to be used. In Linux, these two tasks are usually accomplished using separate tools, although some tools can handle both tasks simultaneously.

🌐 Real World Scenario

When to Create Multiple Partitions

One problem with splitting off lots of separate partitions, particularly for new administrators, is that it can be difficult to settle on appropriate partition sizes. As noted in Table 3.3, the appropriate size of various partitions can vary substantially from one system to another. For instance, a workstation is likely to need a fairly small /var partition (say, 100MB), but a mail or news server might need a /var partition that's gigabytes in size. Guessing wrong isn't fatal, but it is annoying. You'll need to resize your partitions (which is tedious and dangerous) or set up symbolic links between partitions so that subdirectories on one partition can be stored on other partitions.

For this reason, I generally recommend that new Linux administrators try simple partition layouts first. The root (/) partition is required, and swap is a very good idea. Beyond this, /boot can be very helpful on hard disks of more than 8GB with older distributions or BIOSes but is seldom needed with computers or distributions sold since 2000. An appropriate size for /home is often relatively easy for new administrators to guess, so splitting it off generally makes sense. Beyond this, I recommend that new administrators proceed with caution.

As you gain more experience with Linux, you may want to break off other directories into their own partitions on subsequent installations or when upgrading disk hardware. You can use the du command to learn how much space is used by files within any given directory.

Partitioning a Disk

The traditional Linux tool for disk partitioning is called fdisk. This tool's name is short for *fixed disk,* and the name is the same as a DOS and Windows tool that accomplishes the same task. (When I mean to refer to the DOS/Windows tool, I capitalize its name, as in FDISK. The Linux tool's name is always entirely lowercase.) Both DOS's FDISK and Linux's fdisk are text-mode tools to accomplish similar goals, but the two are very different in operational details.

   Although fdisk is the traditional tool, several others exist. One of these that's gaining in popularity is GNU Parted, which can both partition a disk and prepare the partitions for use in a single operation. GNU Parted can also resize several partition types without losing data. Although this operation is risky, dynamic partition resizing can save a lot of time and effort.

Using *fdisk*

To use Linux's fdisk, type the command name followed by the name of the disk device you want to partition, as in **fdisk /dev/hda** to partition the primary master ATA disk. The result is an fdisk prompt. On most modern disks, you'll also see a note telling you that the number of cylinders is greater than 1024:

```
# fdisk /dev/hda

The number of cylinders for this disk is set to 7297.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help):
```

   At the Command (m for help): prompt, you can type commands to accomplish various goals:

Display the current partition table   You may want to begin by displaying the current partition table. To do so, type **p**. If you *only* want to display the current partition table, you can type **fdisk -l /dev/hda** (or whatever the device identifier is) at a command prompt rather than enter fdisk's interactive mode. This command displays the partition table and then exits.

Create a partition   To create a partition, type **n**. The result is a series of prompts asking for information about the partition—whether it should be a primary, extended, or logical partition; the partition's starting cylinder; the partition's ending cylinder or size; and so on. The details of what you're asked depend in part of what's already defined. For instance, fdisk won't ask you

if you want to create an extended partition if one already exists. One fdisk feature you may find odd is that it measures partition start and end points in cylinders, not megabytes. This is a holdover from the CHS measurements used by the *x*86 partition table. In most cases it's not a problem; you can simply pick the default start point and then specify the partition size in megabytes or gigabytes and fdisk will compute the correct end cylinder.

Delete a partition   To delete a partition, type **d**. The program will ask for the partition number, which you must enter.

Change a partition's type   When you create a partition, fdisk assigns it a type code of 0x83, which corresponds to a Linux filesystem. If you want to create a Linux swap partition or a partition for another OS, you can type **t** to change a partition type code. The program then prompts you for a partition number and a type code.

List partition types   Several dozen partition type codes exist, so it's easy to forget what they are. Type **l** (that's a lowercase *L*) at the main fdisk prompt to see a list of the most common ones. You can also get this list by typing **L** when you're prompted for the partition type when you change a partition's type code.

Mark a partition bootable   Some OSs, such as DOS and Windows, rely on their partitions having special bootable flags in order to boot. You can set this flag by typing **a**, whereupon fdisk asks for the partition number.

Get help   Type **m** or **?** to see a summary of the main fdisk commands.

Exit   Linux's fdisk supports two exit modes. First, you can type **q** to exit from the program without saving any changes; anything you do with the program is lost. This option is particularly helpful if you've made a terrible mistake. Second, typing **w** writes your changes to the disk and exits from the program.

As an example, consider deleting a primary, an extended, and a logical partition on a Zip disk and creating a single new one in their place:

```
# fdisk /dev/sda

Command (m for help): p

Disk /dev/sda: 100 MB, 100663296 bytes
4 heads, 48 sectors/track, 1024 cylinders
Units = cylinders of 192 * 512 = 98304 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1               1         510       48936   83  Linux
/dev/sda2             511        1024       49344    5  Extended
/dev/sda5             511        1024       49320   83  Linux
```

```
Command (m for help): d
Partition number (1-5): 5

Command (m for help): d
Partition number (1-5): 2

Command (m for help): d
Selected partition 1

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 4
First cylinder (1-1024, default 1): 1
Last cylinder or +size or +sizeM or +sizeK (1-1024, default 1024): 1024

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

This process begins with a p command to verify that the program is operating on the correct disk. With this information in hand, the three existing partitions are deleted. Note that the first two deletions ask for a partition number, but the third doesn't, because only one partition is left. Once this is done, n is used to create a new primary partition. This example created a partition numbered 4 simply because this is the standard for Zip disks. The task completed, the w command is used to write the changes to disk and exit from the program. The result of this sequence is a Zip disk with a single primary partition (/dev/sda4) marked as holding a Linux filesystem.

Using GNU Parted

GNU Parted (http://www.gnu.org/software/parted/) is a cross-platform partitioning tool—you can use it with non-*x*86 partition tables as well as *x*86 partition tables. It also supports more features than fdisk and is easier to use in some ways. For instance, it measures disk space in megabytes rather than cylinders. GNU Parted also supports dynamic partition resizing for several filesystem types, which can be a great convenience. On the other hand, GNU Parted uses its own way of referring to partitions, which can be confusing. It's also more finicky about mismatched CHS geometries than is fdisk. Although GNU Parted isn't covered on the LPI exam, knowing a bit about it can be handy.

You start GNU Parted much as you start fdisk, by typing its name followed by the device you want to modify, as in **parted /dev/hda** to partition /dev/hda. The result is some brief

introductory text followed by a `(parted)` prompt at which you type commands. Type **?** to see a list of commands, which are multi-character commands similar to Linux shell commands. For instance, `print` displays the current partition table, `mkpart` creates (makes) a partition, `rm` removes a partition, `move` moves a partition, and `resize` changes a partition's size. Some of the more advanced options only work on some filesystem types, such as Linux's native ext2fs and ext3fs and the DOS/Windows standby of FAT. (The next section describes filesystem types in more detail.)

> **WARNING**   Resizing or moving a filesystem can be dangerous. If the resizing code contains a bug or if there's a power failure during the operation, data can be lost. Thus, I strongly recommend you back up any important data before resizing or moving a partition. Also, resizing or moving your boot partition can render the system unbootable until you re-install your boot loader.

Preparing a Partition for Use

Once a partition is created, you must prepare it for use. This process is often called "making a filesystem" or "formatting a partition." It involves writing low-level data structures to disk. Linux can then read and modify these data structures to store files in the partition. You should know something about the common Linux filesystems and know how to use filesystem-creation tools to create them.

> **NOTE**   The word *formatting* is somewhat ambiguous. It can refer to either *low-level formatting,* which involves creating a structure of sectors and tracks on the disk media, or *high-level formatting,* which is creating a filesystem. Hard disks are low-level formatted at the factory and should never need to be low-level formatted again. Floppy disks, though, can be both low- and high-level formatted. The tools described here can high-level format a floppy disk as well as a hard disk. To low-level format a floppy disk, you must use the `fdformat` command, as in **`fdformat /dev/fd0`**. This command cannot be used on a hard disk, though.

Common Filesystem Types

Linux supports quite a few different filesystems, both Linux native and those intended for other OSs. Some of the latter barely work under Linux, and even when they do work reliably, they usually don't support all the features that Linux expects in its native filesystems. Thus, when preparing a Linux system, you'll use one or more of its native filesystems for most or all partitions:

Ext2fs   The *Second Extended File System (ext2fs or ext2)* is the traditional Linux native filesystem. It was created for Linux and was the dominant Linux filesystem throughout the late 1990s. Ext2fs has a reputation as a reliable filesystem. Ext2fs has since been eclipsed by the

other filesystems, but it still has its uses. In particular, ext2fs can be a good choice for a small /boot partition, if you choose to use one, and for small (sub-gigabyte) removable disks. On such small partitions, the size of the journal used by more advanced filesystems can be a real problem, so the non-journaling ext2fs is a better choice. (I describe journaling in more detail shortly.) The ext2 filesystem type code is ext2.

Ext3fs   The *Third Extended File System (ext3fs or ext3)* is basically ext2fs with a journal added. The result is a filesystem that's as reliable as ext2fs but that recovers from power outages and system crashes much more quickly. The ext3 filesystem type code is ext3.

ReiserFS   This filesystem was designed from scratch as a journaling filesystem for Linux and is a popular choice in this role. It's particularly good at handling filesystems with large numbers of small files (say, smaller than about 32KB) because ReiserFS uses various tricks to squeeze the ends of files into each other's unused spaces. This small savings can add up to a large percentage of file sizes when files are small. You can use reiserfs as the type code for this filesystem.

JFS   IBM developed the *Journaled File System (JFS)* for its AIX OS and later re-implemented it on OS/2. The OS/2 version was subsequently donated to Linux. JFS is a technically sophisticated journaling filesystem that might be of particular interest if you're familiar with AIX or OS/2 or want an advanced filesystem to use on a dual-boot system with one of these OSs. As you might expect, this filesystem's type code is jfs.

XFS   Silicon Graphics (SGI) created its *Extents File System (XFS)* for its IRIX OS and, like IBM, later donated the code to Linux. Like JFS, XFS is a very technically sophisticated filesystem. XFS has gained a reputation for robustness, speed, and flexibility on IRIX, but some of the XFS features that make it so flexible on IRIX aren't supported very well under Linux. Use xfs as the type code for this filesystem.

  In practice, most administrators seem to choose ext3fs or ReiserFS as their primary filesystems; however, JFS and XFS also work well, and some administrators prefer them. Hard data on the merits and problems with each filesystem are difficult to come by, and even when they do exist, they're suspect because filesystem performance interacts with so many other factors. For instance, as just noted, ReiserFS can cram more small files into a small space than can other filesystems, but this advantage is not very important if you'll be storing mostly larger files.

> **NOTE**   If you're using a non-*x*86 platform, be sure to check filesystem development on that platform. A filesystem might be speedy and reliable on one CPU but sluggish and unreliable on another.

  In addition to these Linux-native filesystems, you may need to deal with some others from time to time, including the following:

FAT   The *File Allocation Table (FAT)* filesystem is old and primitive. It is, however, ubiquitous. It's the only hard disk filesystem supported by DOS and Windows 9*x*/Me. For this reason, every major OS understands FAT, making it an excellent filesystem for exchanging data on removable

disks. Two major orthogonal variants of FAT exist: It varies in the size of the FAT data structure after which the filesystem is named (12-, 16-, or 32-bit pointers), and it has variants that support long filenames. Linux automatically detects the FAT size, so you shouldn't need to worry about this. To use the original FAT filenames, which are limited to eight characters with an optional three-character extension (the so-called *8.3 filenames*), use the Linux filesystem type code of `msdos`. To use Windows-style long filenames, use the filesystem type code of `vfat`. A Linux-only long filename system, known as `umsdos`, supports additional Linux features—enough that you can actually install Linux on a FAT partition, although this practice isn't recommended except for certain types of emergency disks or to try Linux out on a Windows system.

NTFS   The *New Technology File System (NTFS)* is the preferred filesystem for Windows NT/200*x*/XP. Unfortunately, Linux's NTFS support is rather rudimentary. As of the 2.6.*x* kernel series, Linux can reliably read NTFS and can overwrite existing files, but Linux cannot write new files to an NTFS partition.

> If you must have good NTFS read/write support for a dual-boot system, look into Captive (`http://www.jankratochvil.net/project/captive/`). This is a tool that lets Linux use the NTFS driver delivered with Windows via an interface module. The result is full read/write NTFS support that's fairly reliable. Unfortunately, though, Captive is no longer being maintained.

HFS and HFS+   Apple has long used the *Hierarchical File System (HFS)* with its Mac OS, and Linux provides full read/write HFS support. This support isn't as reliable as Linux's read/write FAT support, though, so you might want to use FAT when exchanging files with Mac users. Apple has extended HFS to better support large hard disks and many Unix-like features with its HFS+ (aka Extended HFS). Linux 2.6.*x* adds limited HFS+ support, but this filesystem is still fairly new in the 2.6.*x* kernels.

ISO-9660   The standard filesystem for CD-ROMs has long been *ISO-9660*. This filesystem comes in several levels. Level 1 is similar to the original FAT in that it supports only 8.3 filenames. Levels 2 and 3 add support for longer 32-character filenames. Linux supports ISO-9660 using its `iso9660` filesystem type code. Linux's ISO-9660 support also works with the *Rock Ridge extensions,* which are a series of extensions to ISO-9660 to enable it to support Unix-style long filenames, permissions, symbolic links, and so on. If a disc includes Rock Ridge extensions, Linux will automatically detect and use them.

Joliet   This filesystem is used much like Rock Ridge, as an extension to ISO-9660, but it's technically a separate filesystem. *Joliet* was created by Microsoft for use by Windows, so it emphasizes Windows filesystem features rather than Unix/Linux filesystem features. Linux supports Joliet as part of its `iso9660` driver; if a disc contains Joliet but not Rock Ridge, Linux uses the Joliet filesystem.

UDF   The *Universal Disc Format (UDF)* is the next-generation filesystem for optical discs. It's commonly used on DVD-ROMs and recordable optical discs. Linux supports it, but read/write UDF support is still in its infancy.

As a practical matter, if you're preparing a hard disk for use with Linux, you should probably use Linux filesystems only. If you're preparing a disk that will be used for a dual-boot configuration, you may want to set aside some partitions for another filesystem type. For removable disks, you'll have to be the judge of what's most appropriate. You might use ext2fs for a Linux-only removable disk, FAT for a cross-platform disk, or ISO-9660 (perhaps with Rock Ridge and Joliet) for a CD-R.

> **NOTE** ISO-9660 and other optical disc filesystems are created with special tools intended just for this purpose. Specifically, mkisofs creates an ISO-9660 filesystem (optionally with Rock Ridge, Joliet, HFS, and UDF components added), while cdrecord writes this image to a blank CD-R.

Creating a Filesystem

Most filesystems, including all Linux-native filesystems, have Linux tools that can create the filesystem on a partition. Typically, these tools have filenames of the form mkfs.*fstype*, where *fstype* is the filesystem type code. These tools can also be called from a front-end tool called mkfs; you pass the filesystem type code to mkfs using its -t option:

```
# mkfs -t ext3 /dev/hda6
```

> **NOTE** For ext2 and ext3 filesystems, the mke2fs program is often used instead of mkfs. The mke2fs program is just another name for mkfs.ext2.

This command creates an ext3 filesystem on /dev/hda6. Depending on the filesystem, the speed of the disk, and the size of the partition, this process can take anywhere from a fraction of a second to a few seconds. Most filesystem-build tools support additional options, some of which can greatly increase the time required to build a filesystem. In particular, the -c option is supported by several filesystems. This option causes the tool to perform a bad block check—every sector in the partition is checked to be sure it can reliably hold data. If it can't, the sector is marked as bad and is not used.

> **WARNING** If you perform a bad block check and find that some sectors are bad, chances are the entire hard disk doesn't have long to live. Sometimes this sort of problem can result from other problems, though, such as bad cables or SCSI termination problems.

Of the common Linux filesystems, ext2fs and ext3fs provide the most options in their mkfs tools. (In fact, these tools are one and the same; the program simply creates a filesystem with a journal when it's called as mkfs.ext3 or when mkfs is called with -t ext3.) You can type **man mkfs.ext2** to learn about these options, most of which deal with obscure and unimportant features. One obscure option that does deserve mention is -m *percent*, which sets the reserved

space percentage. The idea is that you don't want the disk to completely fill up with user files; if the disk starts getting close to full, Linux should report that the disk *is* full before it really is, at least for ordinary users. This gives the root user the ability to log in and create new files, if necessary, to help recover the system.

The ext2fs/ext3fs reserved space percentage defaults to 5 percent, which translates to quite a lot of space on large disks. You might want to reduce this value (say, by passing -m 2 to reduce it to 2 percent) on your root (/) filesystem and perhaps even lower (1 percent or 0 percent) on some, such as /home. Setting -m 0 also makes sense on removable disks, which aren't likely to be critical for system recovery and are likely to be a bit cramped to begin with.

In addition to providing filesystem creation tools for Linux-native filesystems, Linux distributions usually provide such tools for various non-Linux filesystems. The most important of these may be for FAT. The main tool for this task is called mkdosfs, but it's often linked to the mkfs.msdos and mkfs.vfat names, as well. This program can automatically adjust the size of the FAT data structure to 12 or 16 bits depending on the device size. You can override this option with the -F *fat-size* option, where *fat-size* is the FAT size in bits—12, 16, or 32. In fact, this option is *required* if you want to create a FAT-32 partition, which is a practical necessity for any partition over 2GB in size and is usually desirable for partitions over 512MB in size. No special options are required to create a FAT filesystem that can handle Windows-style (VFAT) long filenames; these are created by the OS and require no special options when the filesystem is created.

EXERCISE 3.1

Creating Filesystems

Try creating some filesystems on a spare partition or a removable disk. Even a floppy disk will do, although you won't be able to create journaling filesystems on a floppy disk. The following steps assume you're using a Zip disk, /dev/sda4; change the device specification as necessary. *Be sure to use an empty partition!* Accidentally entering the wrong device filename could wipe out your entire system!

This exercise uses a few commands that are described in more detail in Chapter 4. To create some filesystems, follow these steps:

1. Log in as root.

2. Use **fdisk** to verify the partitions on your target disk by typing **fdisk -l /dev/sda**. You should see a list of partitions, including the one you'll use for your tests.

3. Verify that your test partition is *not* currently mounted. Type **df** to see the currently mounted partitions and verify that /dev/sda4 is not among them.

4. Type **mkfs -t ext2 /dev/sda4**. You should see several lines of status information appear.

5. Type **mount /dev/sda4 /mnt** to mount the new filesystem to /mnt. (You may use another mount point, if you like.)

6. Type **df /mnt** to see basic accounting information on the filesystem. On a 100MB Zip disk, you should see that 95,171 blocks are present, 13 blocks are used, and 90,244 blocks are available. The difference between the present and available blocks is caused by the 5 percent reserved space.

7. Type **umount /mnt** to unmount the filesystem.

8. Type **mkfs -t ext2 -m 0** to create a new ext2 filesystem on the device, but without any reserved space.

9. Repeat steps 5–7. Note that the available space has increased (to 95,158 blocks on a Zip disk). The available space plus the used space should now total the total blocks.

10. Repeat steps 4–7, but use a filesystem type code of ext3 to create a journaling filesystem. (This won't be possible if you use a floppy disk.) Note how much space is consumed by the journal.

11. Repeat steps 4–7, but use another filesystem, such as JFS or ReiserFS. Note how the filesystem creation tools differ in the information they present and in their stated amounts of available space.

Be aware that, because of differences in how filesystems store files and allocate space, a greater amount of available space when a filesystem is created may not translate into a greater capacity to store files.

Creating Swap Space

Some partitions don't hold files. Most notably, Linux can use a *swap partition,* which is a partition that Linux treats as an extension of memory. (Linux can also use a *swap file,* which is a file that works in the same way. Both are examples of *swap space.*) Linux uses the partition type code of 0x82 to identify swap space, but as with other partitions, this code is mostly a convenience to keep other OSs from trying to access Linux swap partitions; Linux uses /etc/fstab to define which partitions to use as swap space, as described in Chapter 4.

WARNING   Solaris for *x*86 also uses a partition type code of 0x82, but in Solaris, this code refers to a Solaris partition. If you dual-boot between Solaris and Linux, this double meaning of the 0x82 partition type code can cause confusion. This is particularly true when installing the OSs. You may need to use Linux's fdisk to temporarily change the partition type codes to keep Linux from trying to use a Solaris partition as swap space or to keep Solaris from trying to interpret Linux swap space as a data partition.

Although swap space doesn't hold a filesystem *per se,* and isn't mounted in the way that filesystem partitions are mounted, swap space does require preparation similar to creation of a filesystem. This task is accomplished with the `mkswap` command, which can generally be used by passing it nothing but the device identifier:

```
# mkswap /dev/hda7
```

This example turns `/dev/hda7` into swap space. To use the swap space, you must activate it with the `swapon` command:

```
# swapon /dev/hda7
```

To permanently activate swap space, you must create an entry for it in `/etc/fstab`, as described in Chapter 4.

Installing Boot Loaders

Hard disks contain more than partitions and their contents. One very small part of the disk is unusually important: The *master boot record (MBR)* contains the partition table and a *boot loader* (also sometimes called a *boot manager*). The boot loader is the software that the BIOS reads and executes when the system begins to boot. The boot loader, in turn, is responsible for loading the Linux kernel into memory and starting it running. Thus, configuring a hard disk (or at least, your boot hard disk) isn't complete until the boot loader is configured. Although Linux distributions provide semi-automated methods of configuring a boot loader during system installation, you may need to know more, particularly if you recompile your kernel or need to set up an advanced configuration—say, one to select between several OSs.

In practice, two boot loaders are important in Linux: the *Linux Loader (LILO)* and the *Grand Unified Boot Loader (GRUB)*. LILO is older and it's slowly being replaced by GRUB as the most common Linux boot loader, but LILO is still quite common.

> LILO and GRUB are both *x*86 boot loaders. Other platforms have their own boot loaders. Some of these are similar to LILO in operation (and may even be called LILO), but they aren't quite identical. You should consult platform-specific documentation if you need to reconfigure a non-*x*86 boot loader. An exception is the AMD64 platform, which can use *x*86 boot loaders.

Boot Loader Principles

The *x*86 boot process can be a bit convoluted, in part because so many options are available. Figure 3.7 depicts a typical configuration, showing a couple of possible boot paths. In both cases, the boot process begins with the BIOS. As described earlier, in "Boot Disks and Geometry Settings," you tell the BIOS which boot device to use—a hard disk, a floppy disk, a CD-ROM

drive, or something else. Assuming you pick a hard disk as the primary boot device, or if higher-priority devices aren't bootable, the BIOS loads code from the MBR. This code is the primary boot loader code. In theory, it could be just about anything, even a complete (if tiny) OS. In practice, the primary boot loader does one of two things:

- It examines the partition table and locates the partition that's marked as bootable. The primary boot loader then loads the boot sector from that partition and executes it. This boot sector contains a secondary boot loader, which continues the process by locating an OS kernel, loading it, and executing it. This option is depicted by the "A" arrows in Figure 3.7.

- It locates an OS kernel, loads it, and executes it directly. This approach bypasses the secondary boot loader entirely, as depicted by the "B" arrow in Figure 3.7.

Traditionally, *x*86 systems running DOS or Windows follow path "A." DOS and Windows 9*x*/Me ship with very simple boot loaders that provide little in the way of options. Windows NT/200*x*/XP provides a boot loader that can provide limited redirection in the second stage of the "A" path.

Linux's boot loaders, LILO and GRUB, are both much more flexible. They support installation in either the MBR or the boot sector of a boot partition. Thus, you can either keep a DOS/Windows-style primary boot loader and direct the system to boot a kernel from a boot sector installation (path "A") or bypass this step and load the kernel straight from the MBR (path "B"). The first option has the advantage that another OS is unlikely to wipe out LILO or GRUB, because it's stored safely in a Linux partition. Windows has a tendency to write its standard MBR boot loader when it's installed, so if you need to re-install Windows on a dual-boot system, this action will wipe out an MBR-based boot loader. If the boot loader is stored in a Linux partition's boot sector, it will remain intact, although Windows might configure the system to bypass it. To reactivate the Linux boot loader, you must use a tool such as the DOS/Windows FDISK to mark the Linux partition as the boot partition.

A drawback of placing LILO or GRUB in a partition's boot sector is that this partition must normally be a primary partition. (An exception is if you're using some other boot loader in the MBR or in another partition. If this third-party boot loader can redirect the boot process to a logical partition, this restriction goes away.) For this reason, many people prefer putting LILO or GRUB in the hard disk's MBR.

In the end, both approaches work, and for a Linux-only installation, the advantages and disadvantages of both approaches are very minor. Some distributions don't give you an option at install time. For them, you should review your boot loader configuration and, when you must add a kernel or otherwise change the boot loader, modify the existing configuration rather than try to create an entirely new one.

F I G U R E 3 . 7   The *x*86 boot system provides several options for redirecting the process, but ultimately an OS kernel is loaded.
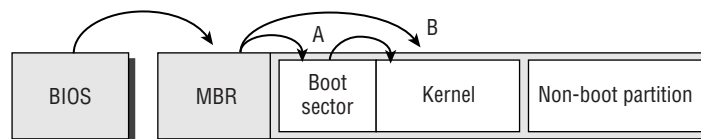
> Both LILO and GRUB can be installed to a floppy disk as well as to a hard disk. Even if you don't want to use a floppy disk as part of your regular boot process, you might want to create an emergency floppy disk with LILO or GRUB. You can then use it to boot Linux if something goes wrong with your regular LILO or GRUB installation.

This description provides a somewhat simplified view of boot loaders. In fact, LILO and GRUB are both much more complex than this. They can redirect the boot process to non-Linux boot sectors and present menus that enable you to boot multiple OSs or multiple Linux kernels. You can chain several boot loaders, including third-party boot loaders such as System Commander or BootMagic. Chaining boot loaders in this way enables you to take advantage of unique features of multiple boot loaders, such as the ability of System Commander to boot several versions of DOS or Windows on a single partition.

Using LILO or GRUB

LILO and GRUB both have their own configuration files. For LILO, the file is `/etc/lilo.conf`; for GRUB, it's either `/boot/grub/grub.conf` or `/boot/grub/menu.lst`, depending on the distribution. Details of these configuration files' layouts and how to edit them are covered in Chapter 6.

If your system is configured and working correctly, you should see a LILO or GRUB display when you boot the computer. Details vary greatly from one system to another, but most modern Linux distributions configure LILO or GRUB to display a menu with all your options. These options may include one or more different Linux kernels, additional dual-boot options, and perhaps options to redirect the boot process to a floppy disk or CD-ROM. You should be able to select an option by using the keyboard's arrow keys and then pressing the Enter key.

Older LILO setups don't present a menu; instead, they display a prompt, usually called `boot:`. You type the name of a kernel or OS definition at this prompt, then press the Enter key. If you don't know the names of the available options, pressing the Tab key displays them.

Both LILO and GRUB are capable of directly booting a Linux kernel or of redirecting the boot process to another OS's partition. GRUB was designed as a cross-platform boot loader, so it includes direct support for some non-Linux OSs, such as FreeBSD; it can load a FreeBSD kernel rather than relying on a FreeBSD boot loader to do that job. (LILO passes this duty off to a FreeBSD boot loader.)

Another important difference between LILO and GRUB is that LILO's options are hard-coded in the boot sector (the MBR or the boot partition's boot sector, depending on its configuration). This means that when you change the LILO configuration, you must re-install the boot sector by typing `lilo` as root. GRUB, by contrast, relies on external configuration files (`/boot/grub/grub.conf` or `/boot/grub/menu.lst`), which it reads whenever its boot sector code is run. Thus, you can alter a GRUB configuration without reinstalling it, just by editing the configuration file. In fact, it's possible to install this file on a FAT or other non-Linux partition, which can sometimes be handy—it enables you to edit the GRUB configuration from a DOS boot floppy, for instance.

Summary

Most Linux tools and procedures provide a layer around the hardware, insulating you from a need to know too many details. Nonetheless, sometimes you've got to dig in and configure hardware directly. BIOS settings can control onboard devices such as ATA controllers and USB ports. Plug-in cards can be configured through jumpers or through Linux tools for managing PnP hardware. Modems have been an important component in the past, and remain important for some users, and they have their own quirks that may need attention. USB and SCSI devices have their own quirks, and USB in particular is quickly evolving.

Hard disks are one class of hardware that's likely to require more attention than most. Specifically, you must know how to create partitions and prepare filesystems on those partitions. These tasks are necessary when you install Linux (although most distributions provide GUI tools to help guide you through this task during installation), when you add a hard disk, or when you reconfigure an existing system. You should also know something about boot managers. These programs help get Linux up and running when you turn on a computer's power, so they really are unusually critical to Linux operation.

Exam Essentials

Summarize BIOS essentials.   The BIOS provides two important functions: First, it configures hardware—both hardware that's built into the motherboard and hardware on many types of plug-in cards. Second, the BIOS begins the computer's boot process, passing control on to the boot loader in the MBR.

Describe what files contain important hardware information.   There are many files under the /proc filesystem. Many of these files have been mentioned throughout this chapter. Familiarize yourself with these files, such as /proc/ioports, /proc/interrupts, /proc/dma, /proc/bus/usb, and others.

Describe how and when to use setserial.   The setserial command is used to view the settings of serial devices. It is also used for changing settings when they are incorrectly guessed or assigned by the BIOS.

Explain Linux's model for managing USB hardware   Linux uses drivers for USB controllers (OHCI, UHCI, and EHCI). These drivers in turn are used by some device-specific drivers (for USB disk devices, for instance) and by programs that access USB hardware via entries in the /proc/bus/usb directory tree.

Summarize the role of other hardware configuration tools.   These tools include ISA-centric tools like isapnp and pnpdump, PCI-centric tools like lspci and setpci, and USB configuration tools like usbmgr and hotplug. Don't overlook their respective configuration files.

Identify SCSI concepts.   Key concepts to familiarize yourself with are SCSI bus sizes and how that impacts the number of devices on a SCSI controller, SCSI IDs (and LUNs), and how attention to SCSI devices is prioritized from SCSI ID 7 down to 0 and then from 15 to 8 (if they are available).

Describe the purpose of disk partitions.   Disk partitions break the disk up into a handful of distinct parts. Each partition can be used by a different OS, can contain a different filesystem, and is isolated from other partitions. These features improve security and safety and can greatly simplify running a multi-OS system.

Summarize important Linux disk partitions.   The most important Linux disk partition is the root (/) partition, which is at the base of the Linux directory tree. Other possible partitions include a swap partition, /home for home directories, /usr for program files, /var for transient system files, /tmp for temporary user files, /boot for the kernel and other critical boot files, and more.

Describe the role of a boot loader and what boot loaders are provided with Linux.   A boot loader takes over from the BIOS, enabling the computer to load an OS kernel or direct the boot process to another (secondary) boot loader. LILO and GRUB are the most important Linux boot loaders.

Review Questions

1. Which device file could be a serial device?

   A. `/dev/ttyS0`

   B. `/dev/ttys0`

   C. `/dev/ttySa`

   D. `/dev/ttysa`

2. What is a common tool for configuring sound cards in Linux?

   A. `soundconfig`

   B. `sndconfig`

   C. `configsound`

   D. `soundconf`

3. What are common IRQs for serial ports? (Select all that apply.)

   A. 1

   B. 3

   C. 4

   D. 16

4. How many devices can be added to an 8-bit SCSI bus, not counting the SCSI host adapter itself?

   A. 7

   B. 8

   C. 15

   D. 16

5. What are the highest- and lowest-priority SCSI IDs on a 16-bit SCSI bus?

   A. 0 and 15, respectively

   B. 7 and 8, respectively

   C. 7 and 0, respectively

   D. 15 and 0, respectively

6. Which files contain essential system information such as IRQs, DMA channels and I/O addresses? (Select all that apply.)

   A. `/proc/ioports`

   B. `/proc/ioaddresses`

   C. `/proc/dma`

   D. `/proc/interrupts`

7. Where are settings for ISA Plug and Play devices stored for the `isapnp` command?

   A. /proc/isa

   B. /etc/isapnp.conf

   C. /etc/pnp.conf

   D. /etc/pnp/isa.conf

8. You've connected two devices to a computer's USB ports: a USB 1.0 mouse and a USB 2.0 printer. Which USB driver modules are required to use both devices at the best possible speeds? (Select all that apply.)

   A. The `usb1.o` and `usb2.o` modules

   B. An OHCI or UHCI module

   C. The `usbcore.o` module

   D. An EHCI module

9. What would be the equivalent RS-232 serial device filename for what Windows would refer to as COM1?

   A. /dev/ttyS0

   B. /dev/ttyS1

   C. /dev/lp0

   D. /dev/lp1

10. USB device drivers are loaded and unloaded by helper applications. Which application may be used by your Linux system? (Select all that apply.)

   A. `usbmgr`

   B. `hotplug`

   C. `usbmanager`

   D. `deviceloader`

11. Typing **fdisk -l /dev/hda** on an *x*86 Linux computer produces a listing of four partitions: /dev/hda1, /dev/hda2, /dev/hda5, and /dev/hda6. Which of the following is true?

   A. The disk contains two primary partitions and two extended partitions.

   B. Either /dev/hda1 or /dev/hda2 is an extended partition.

   C. The partition table is corrupted; there should be a /dev/hda3 and a /dev/hda4 before /dev/hda5.

   D. If you add a /dev/hda3 with `fdisk`, /dev/hda5 will become /dev/hda6 and /dev/hda6 will become /dev/hda7.

12. A new Linux administrator plans to create a system with separate /home, /usr/local, and /etc partitions. Which of the following best describes this configuration?

    A. The system won't boot because /etc contains configuration files necessary to mount non-root partitions.

    B. The system will boot, but /usr/local won't be available because mounted partitions must be mounted directly off their parent partition, not in a subdirectory.

    C. The system will boot only if the /home partition is on a separate physical disk from the /usr/local partition.

    D. The system will boot and operate correctly, provided each partition is large enough for its intended use.

13. Which of the following directories is *most* likely to be placed on its own hard disk partition?

    A. /bin

    B. /sbin

    C. /mnt

    D. /home

14. You discover that an *x*86 hard disk has partitions with type codes of 0x0f, 0x82, and 0x83. Assuming these type codes are accurate, what can you conclude about the disk?

    A. The disk holds a partial or complete Linux system.

    B. The disk holds DOS or Windows 9*x*/Me and Windows NT/200*x*/XP installations.

    C. The disk holds a FreeBSD installation

    D. The disk is corrupt; those partition type codes are incompatible.

15. You run Linux's fdisk and modify your partition layout. Before exiting from the program, though, you realize that you've been working on the wrong disk. What can you do to correct this problem?

    A. Nothing; the damage is done, so you'll have to recover data from a backup.

    B. Type **w** to exit from fdisk without saving changes to disk.

    C. Type **q** to exit from fdisk without saving changes to disk.

    D. Type **u** repeatedly to undo the operations you've made in error.

16. What does the following command accomplish?

    # **mkfs -t ext2 /dev/sda4**

    A. It sets the partition table type code for /dev/sda4 to ext2.

    B. It converts a FAT partition into an ext2fs partition without damaging the partition's existing files.

    C. It creates a new ext2 filesystem on /dev/sda4, overwriting any existing filesystem and data.

    D. Nothing; the -t option isn't valid, and so it causes mkfs to abort its operation.

17. Which of the following best summarizes the differences between DOS's FDISK and Linux's `fdisk`?

    A. Linux's `fdisk` is a simple clone of DOS's FDISK but written to work from Linux rather than from DOS or Windows.

    B. The two are completely independent programs that accomplish similar goals, although Linux's `fdisk` is more flexible.

    C. DOS's FDISK uses GUI controls, whereas Linux's `fdisk` uses a command-line interface, but they have similar functionality.

    D. Despite their similar names, they're completely different tools—DOS's FDISK handles disk partitioning, whereas Linux's `fdisk` formats floppy disks.

18. What mount point should you associate with swap partitions?

    A. `/`

    B. `/swap`

    C. `/boot`

    D. None

19. Which of the following correctly describes a difference between LILO and GRUB?

    A. LILO includes explicit support for FreeBSD kernels; GRUB does not.

    B. LILO can boot Linux kernels through the 2.4.*x* series; GRUB is necessary to boot 2.6.*x* and later kernels.

    C. You must reinstall LILO to the MBR or partition boot sector after changing its configuration; this is unnecessary with GRUB.

    D. LILO can chain load another OS's boot loader, whereas GRUB can boot only Linux kernels.

20. Where should you install GRUB if your system dual-boots Linux and Windows and you don't want Windows to wipe out GRUB if you re-install Windows?

    A. The MBR of the first hard disk

    B. `/var/grub.conf`

    C. The boot sector of a logical Linux partition

    D. The boot sector of a primary Linux partition

Answers to Review Questions

1. A. Only /dev/ttyS0 is acceptable. Serial devices all have a lowercase tty and a capital S followed by a number for their naming convention.

2. B. The sndconfig command is a Red Hat creation for helping to configure sound cards. The other command names are fictional.

3. B, C. IRQs 3 and 4 are common defaults for serial ports. IRQ 1 is reserved for the keyboard, and there is no IRQ 16. IRQs only go from number 0 up to number 15.

4. A. An 8-bit SCSI bus only has 8 SCSI IDs available and 1 is needed for the controller itself. That leaves only 7 more IDs available for devices. Similarly a 16-bit SCSI bus may only have 15 devices.

5. B. Because of historical circumstances, SCSI ID 7 remains the highest priority ID. Priorities go from 7 down to 0 and then start up again at 15 and descend to 8.

6. A, C, D. There is no /proc/ioaddresses file. All other files contain useful information; /proc/ioports holds information on I/O ports, /proc/dma holds information on DMA port usage, and /proc/interrupts holds information on IRQs.

7. B. The isapnp configuration file is /etc/isapnp.conf. The other options are all fictitious files.

8. B, C, D. To use the USB 1.0 mouse, you need an OHCI or UHCI module, depending on the computer's hardware. To use the USB 2.0 printer at full speed, you need the EHCI module. (The printer will work as a USB 1.1 device with an OHCI or UHCI driver, but it will probably print more slowly like this.) The usbcore.o module is at the very base of the USB driver hierarchy; it's always required. There are no standard usb1.o or usb2.o modules.

9. A. /dev/ttyS0 is the only correct answer. /dev/ttyS1 is equivalent to COM2. /dev/lp0 and /dev/lp1 are for parallel ports and are equivalent to LPT1 and LPT2, respectively.

10. A, B. The applications usbmgr and hotplug manage this task. Options C and D do not exist.

11. B. Logical partitions are numbered from 5 and up, and they reside inside an extended partition with a number between 1 and 4. Therefore, one of the first two partitions must be an extended partition that houses partitions 5 and 6. Because logical partitions are numbered starting at 5, their numbers won't change if /dev/hda3 is subsequently added. The disk holds one primary, one extended, and two logical partitions.

12. A. The /etc/fstab file contains the mapping of partitions to mount points, so /etc must be an ordinary directory on the root partition, not on a separate partition. Options B and C describe restrictions that don't exist. Option D would be correct if /etc were not a separate partition.

13. D. The /home directory is frequently placed on its own partition in order to isolate it from the rest of the system and sometimes to enable use of a particular filesystem or filesystem mount options. The /bin and /sbin directories should *never* be split off from the root (/) filesystem because they contain critical executable files that must be accessible in order to do the most basic work, including mounting filesystems. The /mnt directory often contains subdirectories used for mounting floppy disks, CD-ROMs, and other removable media or may be used for this purpose itself. It's seldom used to directly access hard disk partitions, although it can be used for this purpose.

14. A. The 0x0f partition type code is one of two valid partition type codes for an extended partition. (The other is 0x05.) The 0x82 code refers to a Linux swap partition, while 0x83 denotes a Linux filesystem partition. Thus, it appears that this disk holds Linux partitions. Windows 9*x*/Me, Windows NT/200*x*/XP, and FreeBSD all use other partition type codes for their partitions. Partitions exist, in part, to enable different OSs to store their data side-by-side on the same disk, so mixing several partition types (even for different OSs) on one disk does not indicate disk corruption.

15. C. Linux's fdisk doesn't write changes to disk until you exit from the program by typing **w**. Typing **q** exits without writing those changes, so typing **q** in this situation will avert disaster. Typing **w** would be precisely the wrong thing to do. Typing **u** would do nothing useful since it's not an undo command.

16. C. The mkfs command creates a new filesystem, overwriting any existing data and therefore making existing files inaccessible. This command does not set the partition type code in the partition table. The -t ext2 option tells mkfs to create an ext2 filesystem; it's a perfectly valid option.

17. B. Although they have similar names and purposes, Linux's fdisk is not modeled after DOS's FDISK. DOS's FDISK does *not* have GUI controls. Linux's fdisk does *not* format floppy disks.

18. D. Swap partitions aren't mounted in the way filesystems are, so they have no associated mount points.

19. C. Option C correctly describes the requirements for reinstallation of LILO and GRUB after changing their configurations. Option A has it backwards; it's GRUB with explicit support for FreeBSD (and some other OS kernels), not LILO. Neither LILO nor GRUB is limited to booting 2.4.*x* and earlier Linux kernels, so option B is incorrect. Both LILO and GRUB can chain load another boot loader, so option D is incorrect.

20. D. Placing a Linux boot loader, such as LILO or GRUB, in the boot sector of a primary Linux partition will protect that boot loader from being deleted when you re-install Windows. (It may still be deactivated, but can be reactivated with the DOS/Windows FDISK program.) The standard *x*86 MBR boot loader can't redirect the boot process to a logical partition, so option C won't work unless you use a third-party MBR boot loader, in which case it would be wiped out by a Windows re-installation. Placing GRUB in the MBR of the first hard disk would make it vulnerable to damage by Windows re-installation. Although some distributions call their GRUB configuration files grub.conf, this file is stored in /boot/grub, not in /var, and it's not where the main boot loader code itself resides.

Chapter

4

Managing Files and Filesystems

THE FOLLOWING LINUX PROFESSIONAL INSTITUTE OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **1.103.3 Perform basic file management (weight: 3)**
- ✓ **1.104.2 Maintain the integrity of filesystems (weight: 3)**
- ✓ **1.104.3 Control mounting and unmounting filesystems (weight: 3)**
- ✓ **1.104.4 Managing disk quotas (weight: 3)**
- ✓ **1.104.5 Use file permissions to control access to files (weight: 5)**
- ✓ **1.104.6 Manage file ownership (weight: 1)**
- ✓ **1.104.7 Create and change hard and symbolic links (weight: 1)**
- ✓ **1.104.8 Find system files and place files in the correct location (weight: 5)**

Ultimately, Linux is a collection of files stored on your hard disk. Other disk files contain all your user data. For these reasons, being able to manage disk filesystems and the files they contain are important skills for any Linux system administrator. Chapter 3, "Configuring Hardware," described creating disk partitions and preparing filesystems on them. This chapter continues this topic by looking more closely at filesystem maintenance and file management.

> Chapter 3 includes a description of the filesystems that are available for Linux. Although the LPI exam focuses on the Second Extended File System (ext2fs), other filesystems—most notably the Third Extended File System (ext3fs) and ReiserFS, but also the Extents File System (XFS) and the Journaled File System (JFS)—have become dominant in the Linux world.

This chapter begins with an examination of filesystem health—how to track disk usage, tune filesystems for optimal performance, check their internal consistency, and repair simple defects. Assuming a filesystem is in good shape, you must be able to mount it to be able to use it, so that topic is up next. Once a filesystem is mounted, you presumably want to access and manipulate the files it contains, so I describe the commands you can use to do this. As a multi-user OS, Linux provides tools that enable you to restrict *who* may access your files, so I describe the Linux ownership model and the commands that are built upon this model to control file access. Furthermore, Linux provides a system that enables you to restrict how much disk space individual users may consume, so I describe this feature. Finally, this chapter looks at locating files—both the formal description of where certain types of files should reside and the commands you can use to locate files.

Maintaining Filesystem Health

Filesystems can become "sick" in a variety of ways. They can become overloaded with too much data, they can be tuned inappropriately for your system, or they can become corrupted because of buggy drivers, buggy utilities, or hardware errors. Fortunately, Linux provides a variety of utilities that can help you keep an eye on the status of your filesystems, tune their performance, and fix them.

> Many of Linux's filesystem maintenance tools should be run when the filesystem is not mounted. Changes made by maintenance utilities while the filesystem is mounted can confuse the kernel's filesystem drivers, resulting in data corruption. In the following pages, I mention when utilities can and cannot be used with mounted filesystems.

Tuning Filesystems

Filesystems are basically just big data structures—they're a means of storing data on disk in an indexed method that makes it easy to locate the data at a later time. Like all data structures, filesystems include design compromises. For instance, a design feature might enable you to store more small files on disk but it might chew up disk space, thus reducing the total capacity available for storage of larger files. In many cases, you have no choice concerning these compromises; however, some filesystems include tools that enable you to set filesystem options that affect performance. This is particularly true of ext2fs and the related ext3fs. Three tools are particularly important for tuning these filesystems: `dumpe2fs`, `tune2fs`, and `debugfs`. The first of these tools provides information about the filesystem, while the other two enable you to change tuning options.

Obtaining Ext2fs Information

You can learn a lot about your ext2 or ext3 filesystem with the `dumpe2fs` command. This command's syntax is fairly straightforward:

```
dumpe2fs [options] device
```

The `device` is the filesystem device file, such as `/dev/hda2` or `/dev/sdb7`. This command accepts several `options`, most of which are rather obscure. The most important option is probably `-h`, which causes the utility to omit information on group descriptors. (This information is helpful in very advanced filesystem debugging, but not for basic filesystem tuning.) For information on additional options, consult the `man` page for `dumpe2fs`.

Unless you're a filesystem expert and need to debug a corrupted filesystem, you're most likely to want to use `dumpe2fs` with the `-h` option. The result is about three dozen lines of output, each specifying a particular filesystem option, like these:

```
Last mounted on:        <not available>
Filesystem features:    has_journal filetype sparse_super
Filesystem state:       clean
Inode count:            657312
Block count:            1313305
Last checked:           Sun May 30 23:39:16 2004
Check interval:         15552000 (6 months)
```

Some of these options' meanings are fairly self-explanatory; for instance, the filesystem was last checked (with `fsck`, described in "Checking Filesystems") on May 30. Other options are not so obvious; for instance, the `Inode count` line may be puzzling. (It's a count of the number of *inodes* supported by the filesystem. Each inode contains information for one file, so the number of inodes effectively limits the number of files you can store.)

The next two sections describe some of the options you might want to change. For now, you should know that you can retrieve information on how your filesystems are currently configured using `dumpe2fs`. You can then use this information when modifying the configuration; if your current settings seem reasonable, you can leave them alone, but if they seem ill adapted to your configuration, you can change them.

Unlike many low-level disk utilities, you can safely run dumpe2fs on a filesystem that's currently mounted. This can be handy when you're studying your configuration to decide what to modify.

Adjusting Tunable Filesystem Parameters

The tune2fs program enables you to change many of the filesystem parameters that are reported by dumpe2fs. This program's syntax is fairly simple, but it hides a great deal of complexity:

```
tune2fs [options] device
```

The complexity arises because of the large number of *options* that the program accepts. Each feature that tune2fs enables you to adjust requires its own option:

Adjust maximum mount count   Ext2fs and ext3fs require a periodic disk check with fsck. This check is designed to prevent errors from creeping onto the disk undetected. You can adjust the maximum number of times the disk may be mounted without a check with the -c *mounts* option, where *mounts* is the number of mounts. You can trick the system into thinking the filesystem has been mounted a certain number of times with the -C *mounts* option; this sets the mount counter to *mounts*.

Adjust time between checks   Periodic disk checks are required based on time as well as the number of mounts. You can set the time between checks with the -i *interval* option, where *interval* is the maximum time between checks. Normally, *interval* is a number with the character d, w, or m appended, to specify days, weeks, or months, respectively.

Add a journal   The -j option adds a journal to the filesystem, effectively converting an ext2 filesystem into an ext3 filesystem. Journal management is described in more detail shortly, in "Maintaining a Journal."

Set the reserved blocks   The -m *percent* option sets the percentage of disk space that's reserved for use by root. The default value is 5, but this is excessive on large multi-gigabyte hard disks, so you may want to reduce it. You may want to set it to 0 on removable disks intended to store user files. You can also set the reserved space in blocks, rather than by the percentage of disk space, with the -r *blocks* option.

The options described here are the ones that are most likely to be useful. Several other options are available; consult tune2fs's man page for details.

As with most low-level disk utilities, you shouldn't use tune2fs to adjust a mounted filesystem. If you want to adjust a key mounted filesystem, such as your root (/) filesystem, you may need to boot up an emergency disk system, such as the CD-ROM-based Knoppix (http://www.knoppix.org).

Interactively Debugging a Filesystem

In addition to reviewing and changing filesystem flags with dumpe2fs and tune2fs, you can interactively modify a filesystem's features using debugfs. This program provides the abilities of dumpe2fs, tune2fs, and many of Linux's normal file-manipulation tools all rolled into one.

To use the program, type its name followed by the device filename corresponding to the filesystem you want to manipulate. You'll then see the debugfs prompt:

```
# debugfs /dev/hda11
debugfs:
```

You can type commands at this prompt to achieve specific goals:

Display filesystem superblock information   The show_super_stats or stats command produces superblock information, similar to what dumpe2fs displays.

Display inode information   You can display the inode data on a file or directory by typing **stat *filename***, where *filename* is the name of the file.

Undelete a file   You can use debugfs to undelete a file by typing **undelete *inode name***, where *inode* is the inode number of the deleted file and *name* is the filename you want to give to it. (You can use undel in place of undelete if you like.) This facility is of limited utility because you must know the inode number associated with the deleted file. You can obtain a list of deleted inodes by typing **lsdel** or **list_deleted_inodes**, but the list you obtain might not provide enough clues to let you zero in on the correct one.

Extract a file   You can extract a file from the filesystem by typing **write *internal-file external-file***, where *internal-file* is the name of a file in the filesystem you're manipulating and *external-file* is a filename on your main Linux system. This facility can be handy if a filesystem is badly damaged and you want to extract a critical file without mounting the filesystem.

Manipulate files   Most of the commands described later in this chapter in "Managing Files" work within debugfs. You can change your directory with cd, create lines with ln, remove a file with rm, and so on.

Obtain help   Typing **list_requests**, **lr**, **help**, or **?** produces a summary of available commands.

Exit   Typing **quit** exits from the program.

This summary just scratches the surface of debugfs's capabilities. In the hands of an expert, this program can help rescue a badly damaged filesystem or at least extract critical data from it. To learn more, consult the program's man page.

> **WARNING**   Although debugfs is a useful tool, it's a potentially dangerous one. Don't use it on a mounted filesystem, don't use it unless you have to, and be very careful when using it. If in doubt, leave the adjustments to the experts.

Maintaining a Journal

The ext2 filesystem is a traditional filesystem. Although it's a good performer, it suffers from a major limitation: After a power failure, system crash, or other uncontrolled shutdown, the filesystem could be in an inconsistent state. The only way to safely mount the filesystem so that

you're sure its data structures are valid is to perform a full disk check on it, as described in "Checking Filesystems." This task can be handled automatically when the system boots, but it still takes time—probably several minutes, or perhaps even over an hour on a large filesystem or if the computer has many smaller filesystems.

The solution to this problem is to change to a *journaling filesystem*. Such a filesystem maintains a *journal*, which is a data structure that describes pending operations. Prior to writing data to the disk's main data structures, Linux describes what it's about to do in the journal. When the operations are complete, their entries are removed from the journal. Thus, at any given moment the journal should contain a list of disk structures that *might* be undergoing modification. The result is that, in the event of a crash or power failure, the system can examine the journal and check only those data structures described in it. If inconsistencies are found, the system can roll back the changes, returning the disk to a consistent state without checking every data structure in the filesystem. This greatly speeds the disk check process after power failures and system crashes, making journaling filesystems very popular, particularly on servers and other systems that should suffer from minimal downtime.

Four journaling filesystems are available for Linux: ext3fs, ReiserFS, XFS, and JFS. Of these, the last three require little in the way of journal configuration. Ext3fs is a bit different, though; it's basically just ext2fs with a journal added. This fact means that you can add a journal to an ext2 filesystem, converting it into an ext3 filesystem. In fact, this is what the -j option to tune2fs does, as described earlier in "Adjusting Tunable Filesystem Parameters."

> Although using tune2fs on a mounted filesystem is generally inadvisable, it's safe to use its -j option on a mounted filesystem. The result is a file called .journal that holds the journal. If you add a journal to an unmounted filesystem, the journal file will be invisible.

Adding a journal alone won't do much good, though. In order to use a journal, you must mount the filesystem with the correct filesystem type code—ext3 rather than ext2. (The upcoming section "Mounting and Unmounting Filesystems" describes how to do this.)

The journal, like other filesystem features, has its own set of parameters. You can set these with the -J option to tune2fs. In particular, the size=*journal-size* and device=*external-journal* suboptions enable you to set the journal's size and the device on which it's stored. By default, the system creates a journal that's the right size for the filesystem and stores it on the filesystem itself.

Checking Filesystems

Tuning a filesystem is a task you're likely to perform every once in a while—say, when making major changes to an installation. Another task is much more common: checking a filesystem for errors. Bugs, power failures, and mechanical problems can all cause the data structures on a filesystem to become corrupted. The results are sometimes subtle, but if they are left unchecked, they can cause severe data loss. For this reason, Linux includes tools for verifying a filesystem's integrity and for correcting any problems that might exist. The main tool you'll

use for this purpose is called `fsck`. This program is actually a front end to other tools, such as `e2fsck` (aka `fsck.ext2` and `fsck.ext3`). The syntax for `fsck` is as follows:

```
fsck  [-sACVRTNP] [-t fstype] [--] [fsck-options]  filesystems
```

> The LPI objectives emphasize e2fsck rather than fsck, but as fsck is the more general tool that's useful on more filesystems, it's the form described in more detail in this book.

The more common parameters to `fsck` enable you to perform useful actions:

Check all files   The `-A` option causes `fsck` to check all the filesystems marked to be checked in `/etc/fstab`. This option is normally used in system startup scripts.

Progress indication   The `-C` option displays a text-mode progress indicator of the check process. Most filesystem check programs don't support this feature, but `e2fsck` does.

Verbose output   The `-V` option produces verbose output of the check process.

No action   The `-N` option tells `fsck` to display what it would normally do without actually doing it.

Set the filesystem type   Normally, `fsck` determines the filesystem type automatically. You can force the type with the `-t  fstype` flag, though. If used in conjunction with `-A`, this causes the program to check only the specified filesystem types, even if others are marked to be checked. If `fstype` is prefixed with `no`, then all filesystems *except* the specified type are checked.

Filesystem-specific options   Filesystem check programs for specific filesystems often have their own options. The `fsck` command passes options it doesn't understand, or those that follow a double dash (`--`), to the underlying check program. Common options include `-a` or `-p` (perform an automatic check), `-r` (perform an interactive check), and `-f` (force a full filesystem check even if the filesystem initially appears to be clean).

Filesystem list   The final parameter is usually the name of the filesystem or filesystems being checked, such as `/dev/sda6`.

Normally, you run `fsck` with only the filesystem device name, as in **`fsck  /dev/sda6`**. You can add options as needed, however. Check `fsck`'s `man` page for less common options.

> Run fsck *only* on filesystems that are not currently mounted or that are mounted in read-only mode. Changes written to disk during normal read/write operations can confuse fsck and result in filesystem corruption.

Linux runs `fsck` automatically at startup on partitions that are marked for this in `/etc/fstab`, as described later in the section "Permanently Mounting Filesystems." The normal behavior of `e2fsck` causes it to perform just a quick cursory examination of a partition if it's been unmounted cleanly. The result is that the Linux boot process isn't delayed because of a filesystem check unless

the system wasn't shut down properly. A couple of exceptions to this rule exist, however: e2fsck forces a check if the disk has gone longer than a certain amount of time without checks (normally six months) or if the filesystem has been mounted more than a certain number of times since the last check (normally 20). You can change these options using tune2fs, as described earlier in "Adjusting Tunable Filesystem Parameters." Therefore, you will occasionally see automatic filesystem checks of ext2fs and ext3fs partitions even if the system was shut down correctly.

Journaling filesystems do away with filesystem checks at system startup even if the system was not shut down correctly. Nonetheless, these filesystems still require check programs to correct problems introduced by undetected write failures, bugs, hardware problems, and the like. If you encounter odd behavior with a journaling filesystem, you might consider unmounting it and performing a filesystem check—but be sure to read the documentation first. Some Linux distributions do odd things with some journaling filesystem check programs. Most notably, Mandriva uses a symbolic link from /sbin/fsck.reiserfs to /bin/true. This configuration speeds system boot times should ReiserFS partitions be marked for automatic checks, but it can be confusing if you need to manually check the filesystem. If this is the case, run /sbin/reiserfsck to do the job.

Monitoring Disk Use

One common problem with disks is that they can fill up. To avoid this problem, you need tools to tell you how much space your files are consuming. This is the task of the df program and du, whose task is to summarize disk use on a partition-by-partition and directory-by-directory basis, respectively.

Monitoring Disk Use by Partition

The df command's syntax is as follows:

```
df [options] [files]
```

In the simplest case, you can simply type the command name to see a summary of disk space used on all a system's partitions:

```
$ df
Filesystem     1K-blocks      Used Available Use% Mounted on
/dev/sdb10      5859784    4449900   1409884  76% /
/dev/sdb12      2086264     991468   1094796  48% /opt
/dev/hda13      2541468     320928   2220540  13% /usr/local
/dev/hda9      15361340   10174596   5186744  67% /home
/dev/hda10     22699288   13663408   7882820  64% /other/emu
/dev/hda6        101089      22613     74301  24% /boot
/dev/sdb5       1953216    1018752    934464  53% /other/shared
none             256528          0    256528   0% /dev/shm
speaker:/home   6297248    3845900   2451348  62% /speaker/home
//win/music    17156608    8100864   9055744  48% /win/mp3s
```

This output shows the device file associated with the filesystem, the total amount of space on the filesystem, the used space on the filesystem, the free space on the filesystem, the percentage of space that's used, and the *mount point* (the directory from which the filesystem can be accessed). Typically, when used space climbs above about 80 percent, you should consider cleaning up the partition. The appropriate ceiling varies from one system and partition to another, though. The risk is greatest on partitions that hold files that change frequently, and particularly if large files are likely to be created on a partition, even if only temporarily.

You can fine-tune the effects of df by passing it several options. Each option modifies the df output in a specific way:

Include all filesystems   The -a or --all option includes pseudo filesystems with a size of 0 in the output. These filesystems might include /proc, /sys, /proc/bus/usb, and others.

Use scaled units   The -h or --human-readable option causes df to scale and label its units; for instance, instead of reporting a partition as having 5859784 blocks, it reports the size as 5.6G (for 5.6GB). The -H and --si options have a similar effect, but they use power-of-ten (1,000, 1,000,000, and so on) units rather than power-of-two (1,024, 1,048,576, and so on) units. The -k (--kilobytes) and -m (--megabytes) options force output in these units.

Summarize inodes   By default, df summarizes available and used disk space. You can instead receive a report on available and used inodes by passing the -i or --inodes option. This information can be helpful if a partition has very many small files, which can deplete available inodes sooner than they deplete available disk space.

> The -i option works well for ext2, ext3, XFS, and some other filesystems that create a fixed number of inodes when the filesystem is created. Other filesystems, such as ReiserFS, create inodes dynamically, rendering the -i option meaningless.

Local filesystems only   The -l or --local option causes df to omit network filesystems. This can speed up operation.

Display filesystem type   The -T or --print-type option adds the filesystem type to the information df displays.

Limit by filesystem type   The -t *fstype* or --type=*fstype* option displays only information on filesystems of the specified type. The -x *fstype* or --exclude-type=*fstype* option has the opposite effect; it excludes filesystems of the specified type from the report.

This list is incomplete; consult df's man page for details about more options. In addition to these options, you can specify one or more *files* to df. When you do this, the program restricts its report to the filesystem on which the specified file or directory exists. For instance, to learn about the disk space used on the /home partition, you could type **df /home**. Alternatively, you can give a device filename, as in **df /dev/hda9**.

Monitoring Disk Use by Directory

The df command is helpful for finding out which partitions are in danger of becoming over-loaded, but once you've obtained this information, you may need to fine-tune the diagnosis and track down the directories and files that are chewing up disk space. The tool for this task is du, which has a syntax similar to that of df:

```
du [options] [directories]
```

This command searches directories you specify and reports how much disk space each is con-suming. This search is recursive, so you can learn how much space the directory and all its subdi-rectories consume. The result can be a very long listing if you specify directories with many files, but several options can reduce the size of this output. Others can perform other helpful tasks as well:

Summarize files as well as directories   Ordinarily, du reports on the space used by the files in directories but not the space used by individual files. Passing the -a or --all option causes du to report on individual files as well.

Compute a grand total   Adding the -c or --total option causes du to add a grand total to the end of its output.

Use scaled units   The -h or --human-readable option causes du to scale and label its units; for instance, instead of reporting the total disk space used as 5859784 blocks, it reports the size as 5.6G (for 5.6GB). The -H and --si options have a similar effect, but they use power-of-ten (1,000, 1,000,000, and so on) units rather than power-of-two (1,024, 1,048,576, and so on) units. The -k (--kilobytes) and -m (--megabytes) options force output in these units.

Count hard links   Ordinarily, du counts files that appear multiple times as hard links only once. This reflects true disk space used, but sometimes you might want to count each link inde-pendently (for instance, if you're creating a CD-R and the file will actually be stored once for each link). To do so, include the -l (that's a lowercase *L*) or --count-links option.

Limit depth   The --max-depth=*n* option limits the report to *n* levels. (The subdirectories' con-tents are counted even if they aren't reported, though.)

Summarize   If you don't want a line of output for each subdirectory in the tree, pass the -s or --summarize option, which limits the report to those files and directories you specify on the command line. This option is equivalent to --max=depth=0.

Limit to one filesystem   The -x or --one-file-system option limits the report to the current filesystem. If another filesystem is mounted within the tree you want summarized, its contents are not included in the report.

This list is incomplete; you should consult du's man page for information on additional options. As an example of du in action, consider using it to discover which of your users is consuming the most disk space in /home. Chances are you're not concerned with the details of which subdirec-tories within each home directory is using the space, so you'll pass the -s option to the program:

```
# du -s /home/*
12        /home/ellen
```

```
35304      /home/freddie
1760       /home/jennie
12078      /home/jjones
0          /home/lost+found
10110324   /home/mspiggy
```

In this example, the wildcard character (`*`) stands in for all the directories in `/home`, thus producing summaries for all of these subdirectories. (For more on this topic, consult the upcoming section "File Naming and Wildcard Expansion Rules.") Clearly, `mspiggy` (or whoever owns the `/home/mspiggy` directory) is the biggest disk space user—or at least, that directory's contents are consuming the most space. You could investigate further, say by typing **`du -s /home/mspiggy/*`** to learn where the disk space is being used within the `/home/mspiggy` directory. In the case of user files, if this space consumption is a problem, you might want to contact this user instead of trying to clean it up yourself.

> **WARNING**
>
> Many types of files shouldn't simply be deleted. For instance, most program files should be removed via the system's package management system, if you decide to remove them. (This topic is covered in Chapter 2, "Managing Software.") If you're not sure what a file is or how it should be removed, don't delete it—try a Web search, type **`man filename`**, or otherwise research it to figure out what it is.

Mounting and Unmounting Filesystems

Maintaining filesystems is necessary, but the whole reason filesystems exist is to store files—in other words, to be useful. Under Linux, filesystems are most often used by being *mounted*—that is, associated with a directory. This task can be accomplished on a one-time basis by using tools such as `mount` (and then unmounted with `umount`) or persistently across reboots by editing the `/etc/fstab` file.

Temporarily Mounting or Unmounting Filesystems

Linux provides the `mount` command to mount a filesystem to a mount point. The `umount` command reverses this process. (Yes, `umount` is spelled correctly; it's missing the first `n`.) In practice, using these commands are usually not too difficult, but they support a large number of options.

Syntax and Parameters for *mount*

The syntax for `mount` is as follows:

```
mount [-alrsvw] [-t fstype] [-o options] [device]  [mountpoint]
```

Common parameters for mount support a number of features:

Mount all filesystems   The -a parameter causes mount to mount all the filesystems listed in the /etc/fstab file, which specifies the most-used partitions and devices. The upcoming section "Permanently Mounting Filesystems" describes this file's format.

Mount read-only   The -r parameter causes Linux to mount the filesystem read-only, even if it's normally a read/write filesystem.

Verbose output   As with many commands, -v produces verbose output—the program provides comments on operations as they occur.

Mount read/write   The -w parameter causes Linux to attempt to mount the filesystem for both read and write operations. This is the default for most filesystems, but some experimental drivers default to read-only operation.

Filesystem type specification   Use the -t *fstype* parameter to specify the filesystem type. Common filesystem types are ext2 (for ext2fs), ext3 (for ext3fs), reiserfs (for ReiserFS), jfs (for JFS), xfs (for XFS), vfat (for FAT with VFAT long filenames), msdos (for FAT using only short DOS filenames), iso9660 (for CD-ROM filesystems), nfs (for NFS network mounts), smbfs (for SMB/CIFS network shares), and cifs (a newer driver for SMB/CIFS network shares). Linux supports many others. If this parameter is omitted, Linux will attempt to auto-detect the filesystem type.

> Linux requires support in the kernel or as a kernel module to mount a filesystem of a given type. If this support is missing, Linux will refuse to mount the filesystem in question.

Additional options   You can add many options using the -o parameter. Many of these are filesystem specific.

Device   The *device* is the device filename associated with the partition or disk device, such as /dev/hda4, /dev/fd0, or /dev/cdrom. This parameter is usually required, but it may be omitted under some circumstances, as described shortly.

Mount point   The *mountpoint* is the directory to which the device's contents should be attached. As with *device*, it's usually required, but it may be omitted under some circumstances.

The preceding list of mount parameters isn't comprehensive; consult the mount man page for some of the more obscure options. The most common applications of mount use few parameters because Linux generally does a good job of detecting the filesystem type and the default parameters work reasonably well. For instance, consider this example:

```
# mount /dev/sdb7 /mnt/shared
```

This command mounts the contents of /dev/sdb7 on /mnt/shared, auto-detecting the filesystem type and using the default options. Ordinarily, only root may issue a mount command; however, if /etc/fstab specifies the user, users, or owner option, an ordinary user may mount a filesystem using a simplified syntax in which only the device *or* mount point is specified, but not

both. For instance, a user might type **mount /mnt/cdrom** to mount a CD-ROM if /etc/fstab specifies /mnt/cdrom as its mount point and uses the user, users, or owner option.

> Many Linux distributions ship with auto-mounter support, which causes the OS to automatically mount removable media when they're inserted. In GUI environments, a file browser may also open on the inserted disk. In order to eject the disk, the user will need to unmount the filesystem by using umount, as described shortly, or by selecting an option in the desktop environment.

When Linux mounts a filesystem it ordinarily records this fact in /etc/mtab. This file has a format similar to that of /etc/fstab and is stored in /etc, but it's not a configuration file that you should edit. You might examine this file to determine what filesystems are mounted, though. (The df command, described in more detail in "Monitoring Disk Use by Partition," is another way to learn what filesystems are mounted.)

Options for *mount*

When you do need to use special parameters, it's usually to add filesystem-specific options. Table 4.1 summarizes the most important filesystem options. Some of these are meaningful only in the /etc/fstab file.

TABLE 4.1   Important Filesystem Options for the mount Command

Option	Supported Filesystems	Description
defaults	All	Causes the default options for this filesystem to be used. It's used primarily in the /etc/fstab file to ensure that there's an options column in the file.
loop	All	Causes the loopback device for this mount to be used. Allows you to mount a file as if it were a disk partition. For instance, **mount -t vfat -o loop image.img /mnt/image** mounts the file image.img as if it were a disk.
auto or noauto	All	Mounts or does not mount the filesystem at boot time or when root issues the **mount -a** command. The default is auto, but noauto is appropriate for removable media. Used in /etc/fstab.
user or nouser	All	Allows or disallows ordinary users to mount the filesystem. The default is nouser, but user is often appropriate for removable media. Used in /etc/fstab. When included in this file, user allows users to type **mount */mountpoint*** (where */mountpoint* is the assigned mount point) to mount a disk. Only the user who mounted the filesystem may unmount it.

TABLE 4.1   Important Filesystem Options for the mount Command *(continued)*

Option	Supported Filesystems	Description
users	All	Similar to user, except that any user may unmount a filesystem once it's been mounted.
owner	All	Similar to user, except that the user must own the device file. Some distributions, such as Red Hat, assign ownership of some device files (such as /dev/fd0 for the floppy disk) to the console user, so this can be a helpful option.
remount	All	Changes one or more mount options without explicitly unmounting a partition. To use this option, you issue a mount command on an already-mounted filesystem but with remount along with any options you want to change. This feature can be used to enable or disable write access to a partition, for example.
ro	All	Specifies a read-only mount of the filesystem. This is the default for filesystems that include no write access and for some with particularly unreliable write support.
rw	All read/write filesystems	Specifies a read/write mount of the filesystem. This is the default for most read/write filesystems.
uid=*value*	Most filesystems that don't support Unix-style permissions, such as vfat, hpfs, ntfs, and hfs	Sets the owner of all files. For instance, uid=500 sets the owner to whoever has Linux user ID 500. (Check Linux user IDs in the /etc/passwd file.)
gid=*value*	Most filesystems that don't support Unix-style permissions, such as vfat, hpfs, ntfs, and hfs	Works like uid=*value*, but sets the group of all files on the filesystem. You can find group IDs in the /etc/group file.
umask=*value*	Most filesystems that don't support Unix-style permissions, such as vfat, hpfs, ntfs, and hfs	Sets the umask for the permissions on files. *value* is interpreted in binary as bits to be removed from permissions on files. For instance, umask=027 yields permissions of 750, or −rwxr-x---. Used in conjunction with uid=*value* and gid=*value*, this option lets you control who can access files on FAT, HPFS, and many other foreign filesystems.

T A B L E   4 . 1   Important Filesystem Options for the mount Command *(continued)*

Option	Supported Filesystems	Description
conv=*code*	Most filesystems used on Microsoft and Apple OSs: msdos, umsdos, vfat, hpfs, hfs	If *code* is b or binary, Linux doesn't modify the files' contents. If *code* is t or text, Linux auto-converts files between Linux-style and DOS- or Macintosh-style end-of-line characters. If *code* is a or auto, Linux applies the conversion unless the file is a known binary file format. It's usually best to leave this at its default value of binary because file conversions can cause serious problems for some applications and file types.
norock	iso9660	Disables Rock Ridge extensions for ISO-9660 CD-ROMs.
nojoliet	iso9660	Disables Joliet extensions for ISO-9660 CD-ROMs.

Some filesystems support additional options that aren't described here. The man page for mount covers some of these, but you may need to look to the filesystem's documentation for some filesystems and options. This documentation may appear in /usr/src/linux/Documentation/filesystems or /usr/src/linux/fs/*fsname*, where *fsname* is the name of the filesystem.

Using *umount*

The umount command is simpler than mount. The basic umount syntax is as follows:

umount [-afnrv] [-t *fstype*] [*device* | *mountpoint*]

Most of these parameters have meanings similar to their meanings in mount, but some differences deserve mention:

Unmount all   Rather than unmount partitions listed in /etc/fstab, the -a option causes the system to attempt to unmount all the partitions listed in /etc/mtab, the file that holds information on mounted filesystems. On a normally running system, this operation is likely to succeed only partly because it won't be able to unmount some key filesystems, such as the root partition.

Force unmount   You can tell Linux to force an unmount operation that might otherwise fail with the -f option. This feature is sometimes helpful when unmounting NFS mounts shared by servers that have become unreachable.

Fall back to read-only   The -r option tells umount that if it can't unmount a filesystem, it should attempt to remount it in read-only mode.

Unmount partitions of a specific filesystem type   The `-t` `fstype` option tells the system to unmount only partitions of the specified type. You can list multiple filesystem types by separating them with commas.

The device and mount point   You need to specify only the `device` or only the `mountpoint`, not both.

As with `mount`, normal users cannot ordinarily use `umount`. The exception is if the partition or device is listed in `/etc/fstab` and specifies the `user`, `users`, or `owner` option, in which case normal users can unmount the device. (In the case of `user`, only the user who mounted the partition may unmount it; in the case of `owner`, the user issuing the command must also own the device file, as with `mount`.) These options are most useful for removable-media devices.

> **WARNING**   Be cautious when removing floppy disks. Linux caches accesses to floppies, which means that data may not be written to the disk until some time after a write command. Because of this, it's possible to corrupt a floppy by ejecting the disk, even when the drive isn't active. You must *always* issue a umount command before ejecting a mounted floppy disk. This isn't an issue for most non-floppy removable media because Linux can lock their eject mechanisms, preventing this sort of problem. Another way to write the cache to disk is to use the sync command, but because this command does *not* fully unmount a filesystem, it's not really a substitute for umount.

Permanently Mounting Filesystems

The `/etc/fstab` file controls how Linux provides access to disk partitions and removable media devices. Linux supports a unified directory structure in which every disk device (partition or removable disk) is mounted at a particular point in the directory tree. For instance, you might access a floppy disk at `/mnt/floppy`. The root of this tree is accessed from `/`. Directories off this root may be other partitions or disks, or they may be ordinary directories. For instance, `/etc` should be on the same partition as `/`, but many other directories, such as `/home`, may correspond to separate partitions. The `/etc/fstab` file describes how these filesystems are laid out. (The filename `fstab` is an abbreviation for *filesystem table*.)

The `/etc/fstab` file consists of a series of lines that contain six fields each; the fields are separated by one or more spaces or tabs. A line that begins with a hash mark (#) is a comment, and is ignored. Listing 4.1 shows a sample `/etc/fstab` file.

Listing 4.1: Sample *letc/fstab* File

```
#device        mount point   filesystem options        dump fsck
/dev/hda1      /             ext3       defaults         1 1
LABEL=/home    /home         reiserfs   defaults         0 0
/dev/hdb5      /windows      vfat       uid=500,umask=0 0 0
/dev/hdc       /mnt/cdrom    iso9660    users,noauto     0 0
```

```
/dev/fd0       /mnt/floppy  auto      users,noauto    0 0
server:/home /other/home  nfs       users,exec      0 0
//winsrv/shr /other/win   smbfs     users,credentials=/etc/creds 0 0
/dev/hda4    swap         swap      defaults        0 0
```

The meaning of each field in this file is as follows:

Device   The first column specifies the mount device. These are usually device filenames that reference hard disks, floppy drives, and so on. Some distributions, such as Red Hat, have taken to specifying partitions by their labels, as in the LABEL=/home entry in Listing 4.1. When Linux encounters such an entry, it tries to find the partition whose filesystem has the specified name and mount it. This practice can help reduce problems if partition numbers change, but many filesystems lack these labels. It's also possible to list a network drive, as in server:/home, which is the /home export on the computer called server.

Mount point   The second column specifies the mount point; in the unified Linux filesystem, this is where the partition or disk will be mounted. This should usually be an empty directory in another filesystem. The root (/) filesystem is an exception. So is swap space, which is indicated by an entry of swap.

Filesystem type   The filesystem type code is the same as the type code used to mount a filesystem with the mount command. You can use any filesystem type code you can use directly with the mount command. A filesystem type code of auto lets the kernel auto-detect the filesystem type, which can be a convenient option for removable media devices. Auto-detection doesn't work with all filesystems, though.

Mount options   Most filesystems support several mount options, which modify how the kernel treats the filesystem. You may specify multiple mount options, separated by commas. For instance, uid=500,umask=0 for /windows in Listing 4.1 sets the user ID (owner) of all files to 500 and sets the umask to 0. (User IDs and umasks are covered later in this chapter.) Table 4.1 summarizes the most common mount options. Type **man mount** or consult filesystem-specific documentation to learn more.

Backup operation   The next-to-last field contains a 1 if the dump utility should back up a partition or a 0 if it should not. If you never use the dump backup program, this option is essentially meaningless. (The dump program is a common backup tool, but it's by no means the only one.)

Filesystem check order   At boot time, Linux uses the fsck program to check filesystem integrity. The final column specifies the order in which this check occurs. A 0 means that fsck should *not* check a filesystem. Higher numbers represent the check order. The root partition should have a value of 1, and all others that should be checked should have a value of 2. Some filesystems, such as ReiserFS, should not be automatically checked and so should have values of 0.

If you add a new hard disk or have to repartition the one you've got, you'll probably need to modify /etc/fstab. You might also need to edit it to alter some of its options. For instance, setting the user ID or umask on Windows partitions mounted in Linux may be necessary to let ordinary users write to the partition.

🌐 **Real World Scenario**

Managing User-Mountable Media

You may want to give ordinary users the ability to mount certain partitions or removable media, such as floppies, CD-ROMs, and Zip disks. To do so, create an ordinary /etc/fstab entry for the filesystem, but be sure to add the user, users, or owner option to the options column. Table 4.1 describes the differences between these three options. Listing 4.1 shows some examples of user-mountable media: /mnt/cdrom, /mnt/floppy, /other/home, and /other/win. The first two of these are designed for removable media and include the noauto option, which prevents Linux from wasting time trying to mount them when the OS first boots. The second pair of mount points are network file shares that are mounted automatically at boot time; the users option on these lines enables ordinary users to unmount and then remount the filesystem, which might be handy if, say, ordinary users have the ability to shut down the server.

As with any filesystems you want to mount, you must provide mount points—that is, create empty directories—for user-mountable media. Removable media are usually mounted in subdirectories of /mnt or /media, as described later in "Important Directories and Their Contents."

The credentials option for the /other/win mount point in Listing 4.1 deserves greater elaboration. Ordinarily, most SMB/CIFS shares require a username and password as a means of access control. Although you can use the username=*name* and password=*pass* options to smbfs or cifs, these options are undesirable, particularly in /etc/fstab, because they leave the password vulnerable to discovery—anybody who can read /etc/fstab can read the password. The credentials=*file* option provides an alternative—you can use it to point Linux at a file that holds the username and password. This file has labeled lines:

```
username=hschmidt
password=yiW7t9Td
```

Of course, the file you specify (/etc/creds in Listing 4.1) must be well protected—it must be readable only to root and perhaps to the user whose share it describes.

Managing Files

Basic file management is critical to use of any computer. This is particularly true on Unix-like systems, including Linux, because these systems treat almost everything as a file, including most hardware devices and various specialized interfaces. Thus, being able to create, delete, move, rename, and otherwise manipulate files is a basic skill of any Linux user or system administrator.

To begin, you should understand something of the rules that govern filenames and the shortcuts you can use to refer to files. With this information in hand, you can move on to learning how to manipulate files, how to manipulate directories, and how to manage links.

File Naming and Wildcard Expansion Rules

Linux filenames are much like the filenames on any other OS. Every OS has its filename quirks, though, and these differences can be stumbling blocks to those who move between systems—or to those who want to move files between systems.

Linux filenames can contain uppercase or lowercase letters, numbers, and even most punctuation and control characters. To simplify your life and avoid confusion, though, I recommend restricting non-alphanumeric symbols to the dot (.), the dash (-), and the underscore (_). Some programs create backup files that end in the tilde (~), as well. Although Linux filenames can contain spaces, and although such filenames are common in some OSs, they must be escaped on the Linux command line by preceding the space with a backslash (\) or by enclosing the entire filename in quotes ("). This requirement makes spaces a bit awkward in Linux, so most Linux users substitute dashes or occasionally underscores.

A few characters have special meaning and should never be used in filenames. These include the asterisk (*), the question mark (?), the forward slash (/), the backslash (\), and the quotation mark ("). Although you *can* create files that contain all of these characters except for the forward slash (which serves to separate directory elements) by escaping them, they're likely to cause greater confusion than other symbols.

Linux filename length depends on the filesystem in use. On ext2fs, ext3fs, ReiserFS, XFS, and many others, the limit is 255 characters. If you've ever used DOS, you're probably familiar with the *8.3 filename* limit: DOS filenames are restricted to eight characters followed by an optional three-character extension. These two components are separated by a dot. Although one- to four-character extensions are common in Linux, Linux filenames can contain an arbitrary number of dots. In fact, filenames can *begin* with a dot. These so-called *dot files* are hidden from view by most utilities that display files, so they're popular for storing configuration files in your home directory.

> **NOTE** If you access a File Allocation Table (FAT) filesystem on a floppy disk or partition used by DOS, you can do so using any of three filesystem type codes: `msdos`, which limits you to 8.3 filenames; `vfat`, which supports Windows-style long filenames; or `umsdos`, which is a Linux-only extension that supports Linux-style long filenames.

A pair of filenames are particularly special. The filename that consists of a single dot (.) refers to the current directory, while a double dot (..) refers to the parent directory. For instance, if your current directory is `/home/jerry`, `.` refers to that directory and `..` refers to `/home`.

One critical difference between Linux filenames and those of most other OSs is that Linux treats its filenames in a case-sensitive way—`Filename.txt` is different from `filename.txt` or `FILENAME.TXT`. All three files can exist in a single directory. Under Windows, all three filenames refer to the same file. Although Windows 95 and later all retain the case of the filename, they ignore it when you refer to an existing file, and they don't permit files whose names differ only in case to co-exist in a single directory. This difference isn't a major problem for most people who migrate from Windows to Linux, but you should be aware of it. It can also cause problems when you try to read a FAT disk using the Linux `vfat` driver because Linux has to follow the Windows rules when managing files on that disk.

You can use *wildcards* with many commands. A wildcard is a symbol or set of symbols that stand in for other characters. Three classes of wildcards are common in Linux:

?   A question mark (?) stands in for a single character. For instance, b??k matches book, balk, buck, or any other four-letter filename that begins with b and ends with k.

*   An asterisk (*) matches any character or set of characters, including no character. For instance, b*k matches book, balk, and buck just as does b??k. b*k also matches bk, bbk, and backtrack.

Bracketed values   Characters enclosed in square brackets ([]) normally match any character in the set. For instance, b[ao][lo]k matches balk and book but not buck. It's also possible to specify a range of values; for instance, b[a-z]ck matches any back, buck, and other four-letter filenames of this form whose second character is a lowercase letter. This differs from b?ck—because Linux treats filenames in a case-sensitive way, b[a-z]ck doesn't match bAck, although b?ck does.

Wildcards are actually implemented in the shell and passed to the command you call. For instance, if you type **ls b??k** and that wildcard matches the three files balk, book, and buck, the result is precisely as if you'd typed **ls balk book buck**. The process of wildcard expansion is known as *file globbing* or simply *globbing*.

> **WARNING**   The way wildcards are expanded can lead to some undesirable consequences. For instance, suppose you want to copy two files, specified via a wildcard, to another directory but you forget to give the destination directory. The cp command (described shortly) will interpret the command as a request to copy the first of the files over the second.

File Commands

A few file-manipulation commands are extremely important to everyday file operations. These commands enable you to list, copy, move, rename, and delete files.

The *ls* Command

To manipulate files, it's helpful to know what they are. This is the job of the ls command, whose name is short for *list*. The ls command displays the names of files in a directory. Its syntax is simple:

```
ls [options] [files]
```

The command supports a huge number of options; consult ls's man page for details. The most useful options include the following:

Display all files   Normally, ls omits files whose names begin with a dot (.). These dot files are often configuration files that aren't usually of interest. Adding the -a or --all parameter displays dot files.

Color listing   The `--color` option produces a color-coded listing that differentiates directories, symbolic links, and so on by displaying them in different colors. This works at the Linux console, in xterm windows in X, and from some types of remote logins, but some remote login programs don't support color displays.

Display directory names   Normally, if you type a directory name as one of the *files*, `ls` displays the contents of that directory. The same thing happens if a directory name matches a wildcard. Adding the `-d` or `--directory` parameter changes this behavior to list only the directory name, which is sometimes preferable.

Long listing   The `ls` command normally displays filenames only. The `-l` parameter (a lowercase *L*) produces a long listing that includes information such as the file's permission string (described later in "Understanding Permissions"), owner, group, size, and creation date.

Display file type   The `-p` or `--file-type` option appends an indicator code to the end of each name so you know what type of file it is. The meanings are as follows:

/	directory
@	symbolic link
=	socket
|	pipe

Recursive listing   The `-R` or `--recursive` option causes `ls` to display directory contents recursively. That is, if the target directory contains a subdirectory, `ls` displays both the files in the target directory *and* the files in its subdirectory. The result can be a huge listing if a directory has many subdirectories.

Both the *options* list and the *files* list are optional. If you omit the *files* list, `ls` displays the contents of the current directory. You may instead give one or more file or directory names, in which case `ls` displays information on those files or directories, as in this example:

```
$ ls -p /usr /bin/ls
/bin/ls

/usr:
X11R6/   games/                include/  man/    src/
bin/     i386-glibc20-linux/   lib/      merge@  tmp@
doc/     i486-linux-libc5/     libexec/  sbin/
etc/     i586-mandrake-linux/  local/    share/
```

This output shows both the /bin/ls program file and the contents of the /usr directory. The latter consists mainly of subdirectories, but it includes a couple of symbolic links as well. By default, `ls` creates a listing that's sorted by filename, as shown in this example. Note, though, that uppercase letters (as in X11R6) always appear before lowercase letters (as in bin).

One of the most common `ls` options is -1, which creates a listing like this:

```
$ ls -l t*
-rwxr-xr-x  1 rodsmith users      111 Apr 13 13:48  test
-rw-r--r--  1 rodsmith users   176322 Dec 16 09:34  thttpd-2.20b-1.i686.rpm
-rw-r--r--  1 rodsmith users  1838045 Apr 24 18:52  tomsrtbt-1.7.269.tar.gz
-rw-r--r--  1 rodsmith users  3265021 Apr 22 23:46  tripwire.rpm
```

This output includes the permission strings, ownership, file sizes, and file creation dates in addition to the filenames. This example also illustrates the use of the * wildcard, which matches any string—thus, `t*` matches any filename that begins with `t`.

> **NOTE** You can combine multiple options together by merging them with a single preceding dash, as in `ls -lp` to get a long listing that also includes file type codes. This can save a bit of typing compared to the alternative of `ls -l -p`.

The *cp* Command

The `cp` command copies a file. Its basic syntax is as follows:

```
cp [options] source destination
```

The *source* is normally one or more files, and the *destination* may be a file (when the source is a single file) or a directory (when the source is one or more files). When copying to a directory, `cp` preserves the original filename; otherwise, it gives the new file the filename indicated by *destination*. The command supports a large number of options; consult its man page for more information. Some of the more useful options enable you to modify the command's operation in helpful ways:

Force overwrite   The -f or --force option forces the system to overwrite any existing files without prompting.

Interactive mode   The -i or --interactive option causes `cp` to ask you before overwriting any existing files.

Preserve ownership and permissions   Normally, a copied file is owned by the user who issues the `cp` command and uses that account's default permissions. The -p or --preserve option preserves ownership and permissions, if possible.

Recursive copy   If you use the -R or --recursive option and specify a directory as the *source*, the entire directory, including its subdirectories, will be copied. Although -r also performs a recursive copy, its behavior with files other than ordinary files and directories is unspecified. Most `cp` implementations use -r as a synonym for -R, but this behavior isn't guaranteed.

Update copy   The -u or --update option tells `cp` to copy the file only if the original is newer than the target or if the target doesn't exist.

This list of cp options is incomplete but covers the most useful options. Consult the cp man page for information on additional cp options.

As an example, the following command copies the /etc/fstab configuration file to a backup location in /root, but only if the original /etc/fstab is newer than the existing backup:

```
# cp -u /etc/fstab /root/fstab-backup
```

The *mv* Command

The mv command (short for *move*) is commonly used both to move files and directories from one location to another and to rename them. Linux doesn't distinguish between these two types of operations, although many users do. The syntax of mv is similar to that of cp:

```
mv [options] source destination
```

The command takes many of the same *options* as cp does. From the earlier list, --preserve and --recursive don't apply to mv, but the others do.

To move one or more files or directories, specify the files as the *source* and specify a directory or (optionally for a single file move) a filename for the *destination*:

```
$ mv document.sxw important/purchases/
```

This command copies the document.sxw file into the important/purchases subdirectory. If the copy occurs on one low-level filesystem, Linux does the job by rewriting directory entries; the file's data need not be read and rewritten. This makes mv fast. When the target directory is on another partition or disk, though, Linux must read the original file, rewrite it to the new location, and delete the original. This slows down mv. Also, mv can move entire directories within a filesystem but not between filesystems.

The preceding example used a trailing slash (/) on the destination directory. This practice can help avoid problems caused by typos. For instance, if the destination directory were mistyped as important/purchase (missing the final s), mv would move document.sxw into the important directory under the filename purchase. Adding the trailing slash makes it explicit that you intend to move the file into a subdirectory. If it doesn't exist, mv complains, so you're not left with mysterious misnamed files. You can also use the Tab key to avoid problems. When you press Tab in many Linux shells, such as bash, the shell tries to complete the filename automatically, reducing the risk of a typo.

Renaming a file with mv works much like moving a file, except that the source and destination filenames are in the same directory, as shown here:

```
$ mv document.sxw washer-order.sxw
```

This renames `document.sxw` to `washer-order.sxw` in the same directory. You can combine these two forms as well:

```
$ mv document.sxw important/purchases/washer-order.sxw
```

This command simultaneously moves and renames the file.

The *rm* Command

To delete a file, use the `rm` command, whose name is short for *remove*. Its syntax is simple:

```
rm [options] files
```

The `rm` command accepts many of the same *options* as `cp` or `mv`. Of those described with `cp`, `--preserve` and `--update` do not apply to `rm`, but all the others do. With `rm`, `-r` is synonymous with `-R`.

> **WARNING**
>
> By default, Linux doesn't provide any sort of "trash-can" functionality for its `rm` command; once you've deleted a file with `rm`, it's gone and cannot be recovered without retrieving it from a backup or performing low-level disk maintenance (such as with `debugfs`). Therefore, you should be cautious when using `rm`, particularly when you're logged on as root. This is particularly true when you're using the `-R` option—**`rm -R /`** will destroy an entire Linux installation! Many Linux GUI file managers do implement trash-can functionality so that you can easily recover files moved to the trash (assuming you haven't emptied the trash), so you may want to use a file manager for removing files.

The *touch* Command

Linux-native filesystems maintain three time stamps for every file:

- Creation time
- Last modification time
- Last access time

Various programs rely on these time stamps; for instance, the `make` utility (described in Chapter 2) uses the time stamps to determine which source code files must be recompiled if an object file already exists for the source code file. Thus, sometimes you may need to modify the time stamps. This is the job of the `touch` command, which has the following syntax:

```
touch [options] files
```

By default, `touch` sets the modification and access times to the current time. You might use this if, for instance, you want `make` to recompile a particular source code file even though a newer object file exists. If the specified *files* don't already exist, `touch` creates them as empty files. This can be handy if you want to create some dummy files—say, to experiment with other file-manipulation commands.

You can pass various *options* to touch to have it change its behavior:

Change only the access time   The -a or --time=*atime* option causes touch to change the access time alone, not the modification time.

Change only the modification time   The -m or --time=*mtime* option causes touch to change the modification time alone, not the access time.

Do not create file   If you don't want touch to create any files that don't already exist, pass it the -c or --no-create option.

Set the time as specified   The -t *timestamp* option sets the time to the specified *timestamp*. This value is given in the form *MMDDhhmm[[CC]YY][.ss]*, where *MM* is the month, *DD* is the day, *hh* is the hour (on a 24-hour clock), *mm* is the minute, *[CC]YY* is the year (such as 2005 or 05, which are equivalent), and *ss* is the second. Another way to set a particular time is with the -r *reffile* or --reference=*reffile* option, where *reffile* is a file whose time stamp you want to replicate.

Managing Links

In Linux, a *link* is a way to give a file multiple identities, similar to shortcuts in Windows and aliases in Mac OS. Linux employs links to help make files more accessible, to give commands multiple names, to enable programs that look for the same files in different locations to access the same files, and so on. Two types of links exist: *hard links* and *soft links* (aka *symbolic links*). (Their differences are described in more detail shortly.) The ln command creates links. Its syntax is similar to that of cp:

```
ln [options] source link
```

The *source* is the original file, while the *link* is the name of the link you want to create. This command supports options that have several effects:

Remove target files   The -f option causes ln to remove any existing links or files that have the target *link* name.

Create directory hard links   Ordinarily, you can't create hard links to directories. The root user can do so, though, by passing the -d, -F, or --directory option to ln. (Symbolic links to directories are not a problem.)

Create a symbolic link   The ln command creates hard links by default. To create a symbolic link, pass the -s or --symbolic option to the command.

A few other options exist to perform more obscure tasks; consult ln's man page for details. By default, ln creates hard links, which are produced by creating two directory entries that point to the same file (more precisely, the same inode). Both filenames are equally valid and prominent; neither is a "truer" filename than the other, except that one was created first (when creating the file) and the other was created second. To delete the file, you must delete both hard links to the file. Because of the way hard links are created, they must exist on a single low-level filesystem; you can't create a hard link from, say, your root (/) filesystem to a separate filesystem you've mounted on it, such as

your /home filesystem (if it's a separate filesystem). The underlying filesystem must support hard links. All Linux native filesystems support this feature, but some non-Linux filesystems don't.

Symbolic links, by contrast, are special file types. The symbolic link is a separate file whose contents point to the linked-to file. Linux knows to access the linked-to file whenever you try to access the symbolic link, though, so in most respects accessing a symbolic link works just like accessing the original file. Because symbolic links are basically files that contain filenames, they can point across low-level filesystems—you can point from the root (/) filesystem to a file on a separate /home filesystem, for instance. The lookup process for accessing the original file from the link consumes a tiny bit of time, so symbolic link access is slower than hard link access—but not by enough that you'd notice in any but very bizarre conditions or artificial tests. Long directory listings show the linked-to file:

```
$ ls -l alink.sxw
lrwxrwxrwx  1 rodsmith users 8 Dec  2 15:31 alink.sxw -> test.sxw
```

In practice, symbolic links are more common than are links; their disadvantages are minor and the ability to link across filesystems can be important. Linux employs hard links in certain critical system administration tasks. For instance, System V (SysV) startup scripts use symbolic links in runlevel directories, as described in Chapter 6, "The Boot Process and Scripts." Certain commands that have historically been known by multiple names are also often accessible via links. For example, the /sbin/fsck.ext2, /sbin/fsck.ext3, and /sbin/e2fsck programs are usually links (hard links on some systems, symbolic links on others). You can often leave these links alone, but sometimes you must adjust them. Chapter 6 describes changing the SysV startup script links to affect what programs run when the system boots, for instance.

Directory Commands

Most of the commands that apply to files also apply to directories. In particular, ls, mv, touch, and ln all work with directories, with the caveats mentioned earlier. The cp command also works with directories, but only when you use a recursion option, such as -r. A couple of additional commands, mkdir and rmdir, enable you to create and delete directories, respectively.

The *mkdir* Command

The mkdir command creates a directory. This command's official syntax is as follows:

```
mkdir [options] directory-names
```

In most cases, mkdir is used without *options*, but a few are supported:

Set mode   The -m *mode* or --mode=*mode* option causes the new directory to have the specified permission mode, expressed as an octal number. (The upcoming section "Understanding Permissions" describes permission modes.)

Create parent directories   Normally, if you specify the creation of a directory within a directory that doesn't exist, mkdir responds with a No such file or directory error and doesn't create the directory. If you include the -p or --parents option, though, mkdir creates the necessary parent directory.

The *rmdir* Command

The `rmdir` command is the opposite of `mkdir`; it destroys a directory. Its syntax is similar:

```
rmdir [options] directory-names
```

Like `mkdir`, `rmdir` supports few options, the most important of which handle these tasks:

Ignore failures on nonempty directories   Normally, if a directory contains files or other directories, `rmdir` won't delete it and returns an error message. With the `--ignore-fail-on-non-empty` option, `rmdir` still won't delete the directory, but it doesn't return an error message.

Delete tree   The `-p` or `--parents` option causes `rmdir` to delete an entire directory tree. For instance, typing **rmdir -p one/two/three** causes `rmdir` to delete `one/two/three`, then `one/two`, and finally `one`, provided no other files or directories are present.

> When you're deleting an entire directory tree filled with files, `rm -R` is a better choice than `rmdir` because `rm -R` deletes files within the specified directory but `rmdir` doesn't.

Managing File Ownership

Security is an important topic that cuts across many types of commands and Linux subsystems. In the case of files, security is built upon file ownership and file permissions. These two topics are closely intertwined; ownership is meaningless without permissions that use it, and permissions rely upon the existence of ownership.

Ownership is actually two tiered: Each file has an individual owner and a group with which it's associated (sometimes called the group owner, or simply the file's group). Each group can contain an arbitrary number of users, as described in Chapter 8, "Administering the System." The two types of ownership enable you to provide three tiers of permissions to control access to files: by the file's owner, by the file's group, and to all other users. The commands to manage these two types of ownership are similar, but they aren't identical.

Assessing File Ownership

You can learn who owns a file with the `ls` command, which was described earlier. In particular, that command's `-l` option produces a long listing, which includes both ownership and permission information:

```
$ ls -l
total 1141
-rw-r--r--  1 rodsmith users 219648 Mar  8 13:06 4425ch02.doc
-rw-r--r--  1 rodsmith users 942590 Mar  6 23:31 f0201.tif
```

This long listing includes the username of the owner (rodsmith for both files in this example) and the group name of the files' groups (users for both files in this example). The permission string (-rw-r--r-- for both files in this example) is also important for file security, as described later in "Controlling Access to Files."

In most cases, the usernames associated with files are the same as login usernames. Files can, however, be owned by accounts that are not ordinary login accounts. For instance, some servers have accounts of their own, and server-specific files may be owned by these accounts.

If you delete an account, as described in Chapter 8, the account's files don't vanish, but the account name does. Internally, Linux uses numbers rather than names, so you'll see numbers in place of the username and group name in the ls output. Depending on the file, you might want to archive it, reassign ownership to an existing user, or delete it.

Changing a File's Owner

Whenever a file is created, it's assigned an owner. The superuser can change a file's ownership, though, using the chown command, whose syntax is as follows:

chown [*options*] [*newowner*][:*newgroup*] *filenames*

As you might expect, the *newowner* and *newgroup* variables are the new owner and group for the file; you can provide both or omit either, but you can't omit both. For instance, suppose you want to give ownership of a file to sally and the skyhook group:

chown sally:skyhook forward.sxw

> Linux's chown command accepts a dot (.) in place of a colon (:) to delimit the owner and group, at least as of the core file utilities version 5.2.1. The use of a dot has been deprecated, though, meaning that the developers favor the alternative and may eventually eliminate the use of a dot as a feature.

You can use several options with chown, most of which are fairly obscure. One that's most likely to be useful, though, is -R or --recursive, which implements the ownership change on an entire directory tree. Consult the man page for chown for information on additional options.

The chown command may only be used by root. If an ordinary user tries to use it, the result is an Operation not permitted error message.

Changing a File's Group

Both root and ordinary users may run the chgrp command, which changes the group of a file. (Ordinary users may only change a file's group to a group to which the user belongs, though.) This command's syntax is similar, but simpler, to that of chown:

chgrp [*options*] *newgroup* *filenames*

The chgrp command accepts many of the same options as chown, including -R or --recursive. In practice, chgrp provides a subset of the chown functionality, with the important exception that ordinary users can use chgrp.

Controlling Access to Files

The bulk of the complexity in file ownership and permissions is on the permissions end of things. Linux's system of permissions is moderately complex, so understanding how they work is critical to any manipulation of permissions. With the basic information in hand, you can tackle the commands used to change file permissions.

Understanding Permissions

Linux permissions are fairly complex. In addition to providing access control for files, a few special permission bits exist, which provide some unusual features.

The Meanings of Permission Bits

Consider the file access control string that's displayed with the -l option to ls:

```
$ ls -l test
-rwxr-xr-x  1 rodsmith users      111 Apr 13 13:48  test
```

This string (-rwxr-xr-x in this example) is 10 characters in length. The first character has special meaning—it's the *file type code*. The type code determines how Linux will interpret the file—as ordinary data, a directory, or a special file type. Table 4.2 summarizes Linux type codes.

TABLE 4.2   Linux File Type Codes

Code	Meaning
–	Normal data file; may be text, an executable program, graphics, compressed data, or just about any other type of data.
d	Directory; disk directories are files just like any others, but they contain filenames and pointers to disk inodes.
l	Symbolic link; the file contains the name of another file or directory. When Linux accesses the symbolic link, it tries to read the linked-to file.
p	Named pipe; a pipe enables two running Linux programs to communicate with each other. One opens the pipe for reading, and the other opens it for writing, enabling data to be transferred between the programs.

TABLE 4.2   Linux File Type Codes *(continued)*

Code	Meaning
s	Socket; a socket is similar to a named pipe, but it permits network and bidirectional links.
b	Block device; a file that corresponds to a hardware device to and from which data is transferred in blocks of more than one byte. Disk devices (hard disks, floppies, CD-ROMs, and so on) are common block devices.
c	Character device; a file that corresponds to a hardware device to and from which data is transferred in units of one byte. Examples include parallel and RS-232 serial port devices.

The remaining nine characters of the permission string (`rwxr-xr-x` in the example) are broken up into three groups of three characters. The first group controls the file owner's access to the file, the second controls the group's access to the file, and the third controls all other users' access to the file (often referred to as world permissions).

In each of these three cases, the permission string determines the presence or absence of each of three types of access: read, write, and execute. Read and write permissions are fairly self-explanatory, at least for ordinary files. If the execute permission is present, it means that the file may be run as a program. (Of course, this doesn't turn a non-program file into a program; it only means that a user may run a program if it is a program. Setting the execute bit on a non-program file will probably cause no real harm, but it could be confusing.) The absence of the permission is denoted by a dash (`-`) in the permission string. The presence of the permission is indicated by a letter—`r` for read, `w` for write, or `x` for execute.

Thus, the example permission string of `rwxr-xr-x` means that the file's owner, members of the file's group, and all other users can read and execute the file. Only the file's owner has write permission to the file. You can easily exclude those who don't belong to the file's group, or even all but the file's owner, by changing the permission string, as described in "Changing a File's Mode" later in this chapter.

Individual permissions, such as execute access for the file's owner, are often referred to as *permission bits*. This is because Linux encodes this information in binary form. Because it is binary, the permission information can be expressed as a single 9-bit number. This number is usually expressed in octal (base 8) form because a base-8 number is 3 bits in length, which means that the base-8 representation of a permission string is three digits long, one digit for each of the owner, group, and world permissions. The read, write, and execute permissions each correspond to one of these bits. The result is that you can determine owner, group, or world permissions by adding base-8 numbers: 1 for execute permission, 2 for write permission, and 4 for read permission.

Table 4.3 shows some examples of common permissions and their meanings. This table is necessarily incomplete, though; with 9 permission bits, the total number of possible permissions is 2^9, or 512. Most of those possibilities are peculiar, and you're not likely to encounter or create them except by accident.

TABLE 4.3 Example Permissions and Their Likely Uses

Permission string	Octal code	Meaning
rwxrwxrwx	777	Read, write, and execute permissions for all users.
rwxr-xr-x	755	Read and execute permission for all users. The file's owner also has write permission.
rwxr-x---	750	Read and execute permission for the owner and group. The file's owner also has write permission. Users who are not the file's owner or members of the group have no access to the file.
rwx------	700	Read, write, and execute permissions for the file's owner only; all others have no access.
rw-rw-rw-	666	Read and write permissions for all users. No execute permissions to anybody.
rw-rw-r--	664	Read and write permissions to the owner and group. Read-only permission to all others.
rw-rw----	660	Read and write permissions to the owner and group. No world permissions.
rw-r--r--	644	Read and write permissions to the owner. Read-only permission to all others.
rw-r-----	640	Read and write permissions to the owner, and read-only permission to the group. No permission to others.
rw-------	600	Read and write permissions to the owner. No permission to anybody else.
r--------	400	Read permission to the owner. No permission to anybody else.

Execute permission makes sense for ordinary files, but it's meaningless for most other file types, such as device files. Directories, though, make use of the execute bit in another way. When a directory's execute bit is set, that means that the directory's contents may be searched. This is a highly desirable characteristic for directories, so you'll almost never find a directory on which the execute bit is *not* set in conjunction with the read bit.

Directories can be confusing with respect to write permission. Recall that directories are files that are interpreted in a special way. As such, if a user can write to a directory, that user can create, delete, or rename files in the directory, even if the user isn't the owner of those files and does not have permission to write to those files. The *sticky bit* (described shortly, in "Special Permission Bits") can be used to alter this behavior.

Symbolic links are unusual with respect to permissions. This file type always has 777 (`rwxrwxrwx`) permissions, thus granting all users full access to the file. This access applies only to the link file itself, however, not to the linked-to file. In other words, all users can read the contents of the link to discover the name of the file to which it points, but the permissions on the linked-to file determine its file access. Attempting to change the permissions on a symbolic link affects the linked-to file.

Many of the permission rules do not apply to `root`. The superuser can read or write any file on the computer—even files that grant access to nobody (that is, those that have 000 permissions). The superuser still needs an execute bit to be set to run a program file, but the superuser has the power to change the permissions on any file, so this limitation isn't very substantial. Some files may be inaccessible to `root`, but only because of an underlying restriction—for instance, even `root` can't access a hard disk that's not installed in the computer.

Special Permission Bits

A few special permission options are also supported, and they may be indicated by changes to the permission string:

Set user ID (SUID)   The *set user ID (SUID)* option is used in conjunction with executable files, and it tells Linux to run the program with the permissions of whoever owns the file rather than with the permissions of the user who runs the program. For instance, if a file is owned by `root` and has its SUID bit set, the program runs with `root` privileges and can therefore read any file on the computer. Some servers and other system programs run in this way, which is often called SUID `root`. SUID programs are indicated by an `s` in the owner's execute bit position of the permission string, as in `rwsr-xr-x`.

Set group ID (SGID)   The *set group ID (SGID)* option is similar to the SUID option, but it sets the group of the running program to the group of the file. It's indicated by an `s` in the group execute bit position of the permission string, as in `rwxr-sr-x`.

Sticky bit   The sticky bit has changed meaning during the course of Unix history. In modern Linux implementations (and most modern versions of Unix), it's used to protect files from being deleted by those who don't own the files. When this bit is present on a directory, the directory's files can be deleted only by their owners, the directory's owner, or `root`. The sticky bit is indicated by a `t` in the world execute bit position, as in `rwxr-xr-t`.

> **WARNING**   These special permission bits all have security implications. SUID and SGID programs (and particularly SUID `root` programs) are potential security risks. Although some programs must have their SUID bits set to function properly, most don't, and you shouldn't set these bits unless you're certain that doing so is necessary. The sticky bit isn't dangerous in this way, but as it affects who may delete files in a directory, you should consider its effect—or the effect of *not* having it—on directories to which many users should have write access, such as `/tmp`. Typically, such directories have their sticky bits set.

> ### 🌐 Real World Scenario
>
> **Using ACLs**
>
> Unix-style permissions have served Linux well since its creation, and are emphasized on the LPI exam; however, a new and improved permission system is now available. An *access control list (ACL)* is a list of users or groups and the permissions they're given. Linux ACLs, like Linux owner, group, and world permissions, consist of three permission bits, one each for read, write, and execute permissions. ACLs can be assigned by the file's owner to an arbitrary number of users and groups, making ACLs more flexible than Linux permissions, which are limited to groups defined by the system administrator.
>
> ACLs require support in the underlying filesystem. All the major Linux filesystems now support ACLs, but you may need to recompile your kernel (or at least the relevant kernel module) to activate this support.
>
> ACLs require their own commands to set and view. The setfacl command sets an ACL, while the getfacl command displays the ACLs for a file. Consult these commands' man pages for more information.

Changing a File's Mode

You can modify a file's permissions using the chmod command. This command may be issued in many different ways to achieve the same effect. Its basic syntax is as follows:

chmod [*options*] [*mode*[,*mode...*]] *filename...*

The chmod options are similar to those of chown and chgrp. In particular, --recursive (or -R) will change all the files within a directory tree.

Most of the complexity of chmod comes in the specification of the file's mode. There are two basic forms in which you can specify the mode: as an octal number or as a symbolic mode, which is a set of codes related to the string representation of the permissions.

The octal representation of the mode is the same as that described earlier and summarized in Table 4.3. For instance, to change permissions on report.tex to rw-r--r--, you could issue the following command:

```
$ chmod 644 report.tex
```

In addition, you can precede the three digits for the owner, group, and world permissions with another digit that sets special permissions. Three bits are supported (hence values between 0 and 7): adding 4 sets the set user ID (SUID) bit, adding 2 sets the set group ID (SGID) bit, and adding 1 sets the sticky bit. If you omit the first digit (as in the preceding example), Linux clears all three bits. Using four digits causes the first to be interpreted as the special permissions code. For instance, suppose you've created a script called bigprogram in a text editor. You want to

set both SUID and SGID bits (6); to make the script readable, writeable, and executable by the owner (7); to make it readable and executable by the group (5); and to make it completely inaccessible to all others (0). The following commands illustrate how to do this; note the difference in the mode string before and after executing the chmod command:

```
$ ls -l bigprogram
-rw-r--r--    1 rodsmith users    10323 Oct 31 18:58 bigprogram
$ chmod 6750 bigprogram
$ ls -l bigprogram
-rwsr-s---    1 rodsmith users    10323 Oct 31 18:58 bigprogram
```

A symbolic mode, by contrast, consists of three components: a code indicating the permission set you want to modify (the owner, the group, and so on); a symbol indicating whether you want to add, delete, or set the mode equal to the stated value; and a code specifying what the permission should be. Table 4.4 summarizes all these codes. Note that these codes are all case sensitive.

TABLE 4.4 Codes Used in Symbolic Modes

Permission set code	Meaning	Change type code	Meaning	Permission to modify code	Meaning
u	owner	+	add	r	read
g	group	–	remove	w	write
o	world	=	set equal to	x	execute
a	all			X	execute only if file is directory or already has execute permission
				s	SUID or SGID
				t	sticky bit
				u	existing owner's permissions
				g	existing group permissions
				o	existing world permissions

To use symbolic permission settings, you combine one or more of the codes from the first column of Table 4.4 with one symbol from the third column and one or more codes from the fifth column. You can combine multiple settings by separating them by commas. Table 4.5 provides some examples of chmod using symbolic permission settings.

TABLE 4.5   Examples of Symbolic Permissions with chmod

Command	Initial Permissions	End Permissions
chmod a+x bigprogram	rw-r--r--	rwxr-xr-x
chmod ug=rw report.tex	r--------	rw-rw----
chmod o-rwx bigprogram	rwxrwxr-x	rwxrwx---
chmod g=u report.tex	rw-r--r--	rw-rw-r--
chmod g-w,o-rw report.tex	rw-rw-rw-	rw-r-----

EXERCISE 4.1

Modifying Ownership and Permissions

In this exercise, you will experiment with the effect of Linux ownership and permissions on file accessibility. During this exercise, you will need to use three accounts: root and two user accounts, each in a different group. To study these effects, follow these steps:

1. Log in three times using three virtual terminals, once as root, once as user1, and once as user2. (Use usernames appropriate for your system, though. Be sure that user1 and user2 are in different groups.) If you prefer, instead of using virtual terminals, you can open three xterm windows in an X session, using su to acquire each user's privileges.

2. As root, create a scratch directory—say, /tmp/scratch. Type **mkdir /tmp/scratch**.

3. As root, give all users read and write access to the scratch directory by typing **chmod 0777 /tmp/scratch**.

4. In the user1 and user2 login sessions, change to the scratch directory by typing **cd /tmp/scratch**.

5. As user1, copy a short text file to the scratch directory using cp, as in **cp /etc/fstab ./testfile**.

6. As user1, set 0644 (-rw-r--r--) permissions on the file by typing **chmod 0644 testfile**. Type **ls -l** and verify that the permission string in the first column matches this value (-rw-r--r--).

7. As user2, try to access the file by typing **cat testfile**. The file should appear on the screen.

8. As user2, try to change the name of the file by typing **mv testfile changedfile**. The system won't produce any feedback, but if you type **ls**, you'll see that the file's name has changed. Note that user2 doesn't own the file but can rename it because user2 can write to the directory in which the file resides.

9. As user2, try to change the mode of the file by typing **chmod 0600 changedfile**. The system should respond with an `Operation not permitted` error because only the file's owner may change its permissions.

10. As user2, try to delete the file by typing **rm changedfile**. Depending on your configuration, the system may or may not ask for verification, but it should permit the deletion. This is true despite the fact that user2 doesn't own the file because user2 can write to the directory in which the file resides.

11. As user1, repeat step 5 to re-create the test file.

12. As user1, give the file more restrictive permissions by typing **chmod 0640**. Typing **ls -l** should reveal permissions of -rw-r-----, meaning that the file's owner can read and write the file, members of the file's group can read it, and other users are given no access.

13. As user2, repeat steps 7–10. The cat operation should fail with a `Permission denied` error, but steps 8–10 should produce the same results as they did the first time around. (If the cat operation succeeded, then either user2 belongs to the file's group or the file's mode is set incorrectly.)

14. Log out of the user1 and user2 accounts.

15. As root, type **rm -r /tmp/scratch** to delete the scratch directory and its contents.

If you like, you can perform tests with more file permission modes and other file-manipulation commands before step 14.

As a general rule, symbolic permissions are most useful when you want to make a simple change (such as adding execute or write permissions to one or more class of users) or when you want to make similar changes to many files without affecting their other permissions (for instance, adding write permissions without affecting execute permissions). Octal permissions are most useful when you want to set some specific absolute permission, such as rw-r--r-- (644). In any event, a system administrator should be familiar with both methods of setting permissions.

A file's owner and root are the only users who may adjust a file's permissions. Even if other users have write access to a directory in which a file resides and write access to the file itself, they may not change the file's permissions (but they may modify or even delete the file). To understand why this is so, you need to know that the file permissions are stored as part of the file's inode, which isn't part of the directory entry. Read/write access to the directory entry, or even the file itself, doesn't give a user the right to change the inode structures (except indirectly—for instance, if a write changes the file's size or a file deletion eliminates the need for the inode).

Setting the Default Mode and Group

When a user creates a file, that file has default ownership and permissions. The default owner is, understandably, the user who created the file. The default group is the user's primary group. The default permissions are configurable. These are defined by the *user mask (umask)*, which is set by the `umask` command. This command takes as input an octal value that represents the bits to be removed from 777 permissions for directories, or from 666 permissions for files, when a new file or directory is created. Table 4.6 summarizes the effect of several possible umask values.

T A B L E   4 . 6   Sample Umask Values and Their Effects

Umask	Created Files	Created Directories
000	666 (rw-rw-rw-)	777 (rwxrwxrwx)
002	664 (rw-rw-r--)	775 (rwxrwxr-x)
022	644 (rw-r--r--)	755 (rwxr-xr-x)
027	640 (rw-r-----)	750 (rwxr-x---)
077	600 (rw-------)	700 (rwx------)
277	400 (r--------)	500 (r-x------)

Note that the umask isn't a simple subtraction from the values of 777 or 666; it's a bit-wise removal. Any bit that's set in the umask is removed from the final permission for new files, but if a bit isn't set (as in the execute bit in ordinary files), its specification in the umask doesn't do any harm. For instance, consider the 7 values in several entries of Table 4.6's Umask column. This corresponds to a binary value of 111. An ordinary file might have `rw-` (110) permissions, but applying the umask's 7 (111) eliminates 1 values but doesn't touch 0 values, thus producing a 000 (binary) value—that is, `---` permissions, expressed symbolically.

Ordinary users can enter the `umask` command to change the permissions on new files they create. The superuser can also modify the default setting for all users by modifying a system configuration file. Typically, `/etc/profile` contains one or more `umask` commands. Setting the umask in `/etc/profile` might or might not actually have an effect because it can be overridden at other points, such as a user's own configuration files. Nonetheless, setting the umask in `/etc/profile` or other system files can be a useful procedure if you want to change the default system policy. Most Linux distributions use a default umask of 002 or 022.

To find what the current umask is, type **umask** alone, without any parameters. Typing **umask -S** produces the umask expressed symbolically rather than in octal form. You may also specify a umask in this way when you want to change it, but in this case, you specify the bits that you *do* want set. For instance, **umask u=rwx,g=rx,o=rx** is equivalent to **umask 022**.

In addition to setting the default mask with umask, users can change their default group with newgrp, as in **newgrp skyhook** to create new files with the group set to the skyhook group. To use this command, the user must be a member of the specified group. The newgrp command also accepts the -1 parameter, as in **newgrp -1 skyhook**, which reinitializes the environment as if the user had just logged in.

Changing File Attributes

Some filesystems support attributes in addition to those described in the preceding sections. In particular, most Linux native filesystems support several attributes that you can adjust with the chattr command:

Append only   The a attribute sets append mode, which disables write access to the file except for appending data. This can be a security feature to prevent accidental or malicious changes to files that record data, such as log files.

Compressed   The c attribute causes the kernel to automatically compress data written to the file and uncompress it when it's read back.

Immutable   The i flag makes a file immutable, which goes a step beyond simply disabling write access to the file. The file cannot be deleted, links to it cannot be created, and the file cannot be renamed.

Data journaling   The j flag tells the kernel to journal all data written to the file. This improves recoverability of data written to the file after a system crash but can slow performance. This flag has no effect on ext2 filesystems.

Secure deletion   Ordinarily, when you delete a file, its directory entry is removed and its inode is marked as being available for recycling. The data blocks that make up the bulk of the file are not erased. Setting the s flag changes this behavior; when the file is deleted, the kernel zeros its data blocks, which might be desirable for files that contain sensitive data.

No tail-merging   Tail-merging is a process in which small pieces of data at the ends of files that don't fill a complete block are merged with similar pieces of data from other files. The result is reduced disk space consumption, particularly when you store many small files rather than a few big ones. Setting the t flag disables this behavior, which is desirable if the filesystem will be read by certain non-kernel drivers, such as those that are part of the Grand Unified Boot Loader (GRUB).

No access time updates   If you set the A attribute, Linux won't update the access time stamp when you access a file. This can reduce disk input/output, which is particularly helpful for saving battery life on laptops.

This list of attributes is incomplete but includes the most useful options; consult the man page for chattr for more flags. You set the options you want using the minus (-), plus (+), or equal (=) symbols to remove an option from an existing set, add an option to an existing set, or set a precise set of options (overwriting any that already exist), respectively. For instance, to add the immutable flag to the important.txt file, you'd enter the following command:

```
# chattr +i important.txt
```

The result is that you will then be unable to delete the file, even as `root`. To delete the file, you must first remove the immutable flag:

```
# chattr -i important.txt
```

Managing Disk Quotas

Just one or two users of a multi-user system can cause serious problems for others by consuming too much disk space. If a single user creates huge files (say, multimedia recordings), those files can prevent other users from creating their own files. To help manage this situation, Linux supports *disk quotas*—limits enforced by the OS on how many files or how much disk space a single user may consume. The Linux quota system supports quotas both for individual users and for Linux groups.

Enabling Quota Support

Quotas require support in both the kernel for the filesystem being used and various user-space utilities. As of the early 2.6.*x* kernels, the ext2fs, ext3fs, and ReiserFS filesystems support quotas, but you must explicitly enable support via the Quota Support kernel option in the filesystem area when recompiling your kernel. Many distributions ship with this support precompiled, so recompiling your kernel may not be necessary, but you should be aware of this option if you do recompile your kernel.

Two general quota support systems are available for Linux. The first was used through the 2.4.*x* kernels and is referred to as the quota v1 support. The second was added with the 2.6.*x* kernel series and is referred to as the quota v2 system. This description applies to the latter system, but the former works in a similar way.

Outside of the kernel, you need support tools to use quotas. For the quota v2 system, this package is usually called `quota`, and it installs a number of utilities, configuration files, SysV startup scripts, and so on.

> **NOTE** You can install the support software from source code, if you like; however, this job is handled most easily using a package for your distribution. This description assumes that you install the software in this way. If you don't, you may need to create SysV or local startup scripts to initialize the quota support when you boot your computer. The Quota Mini-HOWTO, at `http://en.tldp.org/HOWTO/Quota.html`, provides details of how to do this.

You must modify your `/etc/fstab` entries for any partitions on which you want to use the quota support. In particular, you must add the `usrquota` filesystem mount option to employ user quotas and the `grpquota` option to use group quotas. Entries that are so configured resemble the following:

```
/dev/hdc5  /home  ext3  usrquota,grpquota  1  1
```

This line activates both user and group quota support for the /dev/hdc5 partition, which is mounted at /home. Of course, you can add other options if you like.

Depending on your distribution, you may need to configure the quota package's SysV startup scripts to run when the system boots. Chapter 5 describes SysV startup script management in detail. Typically, you'll type a command such as **chkconfig quota on**; however, you should check on the SysV scripts installed by your distribution's quota package. Some distributions require use of commands other than chkconfig to do this task, as described in Chapter 6. Whatever its details, this startup script runs the quotaon command, which activates the quota support.

After installing software and making configuration file changes, you must activate the systems. The simplest way to do this is to reboot the computer, and this step is necessary if you had to recompile your kernel to add quota support directly into the kernel. If you didn't do this, though, you should be able to get by with less disruptive measures: using modprobe to install the kernel module, if necessary; running the SysV startup script for the quota tools; and remounting the filesystems on which you intend to use quotas by typing **mount -o remount */mount-point***, where */mount-point* is the mount point in question.

Setting Quotas for Users

At this point, quota support should be fully active on your computer, but the quotas themselves are not set. You can set the quotas by using edquota, which starts the Vi editor (described in Chapter 1, "Linux Command-Line Tools") on a temporary configuration file (/etc/quotatab) that controls quotas for the user you specify. When you exit from the utility, edquota uses the temporary configuration file to write the quota information to low-level disk data structures that control the kernel's quota mechanisms. For instance, you might type **edquota sally** to edit sally's quotas. The contents of the editor will show the current quota information:

```
Quotas for user sally:
/dev/hdc5: blocks in use: 3209, limits (soft = 5000, hard = 6500)
          inodes in use: 403, limits (soft = 1000, hard = 1500)
```

The temporary configuration file provides information on both the number of disk blocks in use and the number of inodes in use. (Each file or symbolic link consumes a single inode, so the inode limits are effectively limits on the number of files a user may own. Disk blocks vary in size depending on the filesystem and filesystem creation options, but they typically range from 512 bytes to 8KB.) Changing the use information has no effect, but you can alter the soft and hard limits for both blocks and inodes. The hard limit is the maximum number of blocks or inodes that the user may consume; the kernel will not permit a user to surpass these limits. Soft limits are somewhat less stringent; users may temporarily exceed soft limit values, but when they do so, the system issues warnings. Soft limits also interact with a grace period; if the soft quota limit is exceeded for longer than the grace period, the kernel begins treating it like a hard limit and refuses to allow the user to create more files. You can set the grace period by using edquota with its -t option, as in **edquota -t**. Grace periods are set on a per-filesystem basis rather than a per-user basis.

A few more quota-related commands are useful. The first is `quotacheck`, which verifies and updates quota information on quota-enabled disks. This command is normally run as part of the `quota` package's SysV startup script, but you may want to run it periodically (say, once a week) as a `cron` job. (Chapter 5 describes `cron` jobs.) Although theoretically not necessary if everything works correctly, `quotacheck` ensures that quota accounting doesn't become inaccurate. The second useful auxiliary quota command is `repquota`, which summarizes the quota information on the filesystem you specify or on all filesystems if you pass it the `-a` option. This tool can be very helpful in keeping track of disk usage. The `quota` command has a similar effect. The `quota` tool takes a number of options to have them modify their outputs. For instance, `-g` displays group quotas, `-l` omits NFS mounts, and `-q` limits output to filesystems on which usage is over the limit. Consult `quota`'s `man` page for still more obscure options.

Locating Files

Maintaining your filesystems in perfect health, setting permissions, and so on is all pointless if you can't find your files. For this reason, Linux provides several tools to help you locate the files you need to use. The first of these tools is actually a standard for where files are located; with the right knowledge, you may be able to find files without the use of any specialized programs. The second class of tools is just such specialized programs, which search a directory tree or a database for files that meet whatever criteria you specify.

The FHS

Linux's placement of files is derived from 30 years of Unix history. Given that fact, the structure is remarkably simple and coherent, but it's easy for a new administrator to become confused. Some directories seem, on the surface, to fulfill similar or even identical roles, but in fact there are subtle but important differences. This section describes the Linux directory layout standards, presents an overview of what goes where, and explains the use of the `du` utility, which is useful in learning where your disk space is being used.

The FSSTND and FHS

Although Linux draws heavily on Unix, Unix's long history has led to numerous splits and variants, starting with the Berkeley Standard Distribution (BSD), which was originally a set of patches and extensions to AT&T's original Unix code. As a result of these schisms within the Unix community, early Linux distributions didn't always follow patterns that were identical to each other. The result was a great deal of confusion. This problem was quite severe early in Linux's history, and it threatened to split the Linux community into factions. Various measures were taken to combat this problem, one of which was the development of the *Filesystem Standard (FSSTND),* which was first released in early 1994. The FSSTND standardized several specific features, including:

▪ Standardized the programs that reside in `/bin` and `/usr/bin`. Differences on this score caused problems when scripts referred to files in one location or the other.

- Specified that executable files should not reside in /etc, as had previously been common.
- Removed changeable files from the /usr directory tree, enabling it to be mounted read-only, a useful security measure.

There have been three major versions of FSSTND: 1.0, 1.1, and 1.2. FSSTND began to rein in some of the chaos in the Linux world in 1994. By 1995, however, FSSTND's limitations were themselves becoming apparent. Thus, a new standard was developed: the *Filesystem Hierarchy Standard (FHS)*. This new standard is based on FSSTND but extends it substantially. The FHS was created in conjunction with developers of some non-Linux Unix-like OSs, for instance. For this reason, the FHS is more than a Linux standard; it may be used to define the layout of files on other Unix-like OSs.

One important distinction made by the FHS is the one between *shareable files* and *unshareable files*. Shareable files may be reasonably shared between computers, such as user data files and even program binary files. (Of course, you don't need to share such files, but you *may* do so.) If files are shared, they're normally shared through an NFS server, described briefly in Chapter 10, "Managing Servers." Unshareable files contain system-specific information, such as configuration files. For instance, you're not likely to want to share a server's configuration file between computers.

A second important distinction used in the FHS is the one between *static files* and *variable files*. The former don't normally change except through direct intervention by the system administrator. Most program executables are good examples of static files. Variable files may be changed by users, automated scripts, servers, or the like. For instance, users' home directories and mail queues are composed of variable files. The FHS tries to isolate each directory into one cell of this 2×2 (shareable/unshareable × static/variable) matrix. Figure 4.1 illustrates these relationships. Some directories are mixed, but in these cases, the FHS tries to specify the status of particular subdirectories. For instance, /var is variable, and it contains some shareable and some unshareable subdirectories, as shown in Figure 4.1.

Like the FSSTND, the FHS comes in numbered versions. Version 2.3, the latest version as I write, was released in January 2004. The URL for FHS's official web page is http://www.pathname.com/fhs/.

FIGURE 4.1   The FHS attempts to fit each important directory in one cell of a 4×4 matrix.

	Shareable	Unshareable
Static	/usr /opt	/etc /boot
Variable	/home /var/mail	/var/run /var/lock

Important Directories and Their Contents

The FHS defines some directories very precisely, but details for others are left unresolved. For instance, users' files normally go in the /home directory, but you may have reason to call this something else or to use two or more separate directories for users' files. Overall, the most common directories defined by the FHS or used by convention are the following:

/ Every Linux filesystem traces its roots to a single directory, known as / (pronounced, and often referred to, as the *root filesystem* or *root directory*). All other directories branch off of this one. Linux doesn't use drive letters; instead, every partition or removable disk is mounted at a mount point within another partition (/ or something else). Certain critical subdirectories, such as /etc and /sbin, must reside on the root partition, but others can optionally be on separate partitions. Do not confuse the root directory with the /root directory, described shortly.

/boot The /boot directory contains static and unshareable files related to the initial booting of the computer. Higher-level startup and configuration files reside in another directory, /etc. Some systems impose particular limits on /boot. For instance, older *x*86 BIOSes and older versions of the Linux Loader (LILO) may require that /boot reside below the 1,024th cylinder of the hard disk. These requirements sometimes, but not always, necessitate that the /boot directory be a separate partition.

/bin This directory contains certain critical executable files, such as ls, cp, and mount. These commands are accessible to all users and constitute the most important commands that ordinary users might issue. You won't normally find commands for big application programs in /bin (although the Vi editor is located here). The /bin directory contains static files. Although in some sense the /bin files are shareable, because they're so important to the basic operation of a computer, the directory is almost never shared—any potential clients must have their own local /bin directories.

/sbin This directory is similar to /bin, but it contains programs that are normally run only by the system administrator—tools like fdisk and e2fsck. Therefore, it's static and theoretically shareable, but in practice, it makes no sense to share it.

/lib This directory is similar to /bin and /sbin, but it contains program libraries, which are made up of code that's shared across many programs and stored in separate files to save disk space and RAM. The /lib/modules subdirectory contains kernel modules—drivers that can be loaded and unloaded as required. Like /bin and /sbin, /lib is static and theoretically shareable, although it's not shared in practice.

/usr This directory hosts the bulk of a Linux computer's programs. Its contents are shareable and static, so it can be mounted read-only and may be shared with other Linux systems. For these reasons, many administrators split /usr off into a separate partition, although this isn't required. Some subdirectories of /usr are similar to their namesakes in the root directory (such as /usr/bin and /usr/lib), but they contain programs and libraries that aren't absolutely critical to the basic functioning of the computer.

/usr/local This directory contains subdirectories that mirror the organization of /usr, such as /usr/local/bin and /usr/local/lib. The /usr/local directory hosts files that a system

administrator installs locally—for instance, packages that are compiled on the target computer itself. The idea is to have an area that's safe from automatic software upgrades when the OS as a whole is upgraded. Immediately after Linux is installed, `/usr/local` should be empty except for some "stub" subdirectories. Some system administrators split this off into its own partition to protect it from OS reinstallation procedures that might erase the parent partition.

/usr/X11R6   This directory houses files related to the X Window System (X for short), Linux's GUI environment. Like `/usr/local`, this directory contains subdirectories similar to those in `/usr` itself, such as `/usr/X11R6/bin` and `/usr/X11R6/lib`.

/opt   This directory is similar to `/usr/local` in many ways, but it is intended for ready-made packages that don't ship with the OS, like commercial word processors or games. Typically, these programs reside in subdirectories in `/opt` named after themselves, such as `/opt/applix`. The `/opt` directory is static and shareable. Some system administrators break it into a separate partition or make it a symbolic link to a subdirectory of `/usr/local` and make that a separate partition.

/home   This directory contains users' data, and it is shareable and variable. Although the `/home` directory is considered optional in FHS, in practice it's a matter of the *name* being optional. For instance, if you add a new disk to support additional users, you might leave the existing `/home` directory intact and create a new `/home2` directory to house the new users. The `/home` directory often resides on its own partition.

/root   This is the home directory for the `root` user. Because the `root` account is so critical and system specific, this variable directory isn't really shareable.

/var   This directory contains transient files of various types—system log files, print spool files, mail and news files, and so on. As such, the directory's contents are variable. Some subdirectories are shareable, but others are not. Many system administrators put `/var` in its own partition, particularly on systems that see a lot of activity in `/var`, like major Usenet news or mail servers.

/tmp   Many programs need to create temporary (hence variable) files, and the usual place to do so is in `/tmp`. Most distributions include routines that clean out this directory periodically, and sometimes wipe the directory clean at bootup. The `/tmp` directory is seldom shared. Some administrators create a separate `/tmp` partition to prevent runaway processes from causing problems on the root filesystem when processes create too-large temporary files. A similar directory exists as part of the `/var` directory tree (`/var/tmp`).

/mnt   Linux mounts removable-media devices within its normal directory structure, and `/mnt` is provided for this purpose. Some distributions create subdirectories within `/mnt`, such as `/mnt/floppy` and `/mnt/cdrom`, to function as mount points. Others use `/mnt` directly or even use separate mount points off /, such as `/floppy` and `/cdrom`. The FHS mentions only `/mnt`; it doesn't specify how it's to be used. Specific media mounted in `/mnt` may be either static or variable. As a general rule, these directories are shareable.

/media   This directory is an optional part of the FHS. It's like `/mnt`, but it should contain subdirectories for specific media types, such as `/media/floppy` and `/media/cdrom`. A few distributions, such as SuSE, now use `/media` subdirectories as the default mount points for common removable disk types.

/dev   Because Linux treats most hardware devices as if they were files, the OS must have a location in its filesystem where these device files reside. The /dev directory is that place. It contains a large number of files that function as hardware interfaces. If a user has sufficient privileges, that user may access the device hardware by reading from and writing to the associated device file. The Linux kernel supports a device filesystem that enables /dev to be an automatically created *virtual filesystem*—the kernel and support tools create /dev entries on-the-fly to accommodate the needs of specific drivers. A few distributions, such as Gentoo, use this feature by default, but many others don't.

/proc   This is an unusual directory because it doesn't correspond to a regular directory or partition. Instead, it's a virtual filesystem that's created dynamically by Linux to provide access to certain types of hardware information that's not accessible via /dev. For instance, if you type **cat /proc/cpuinfo**, the system responds by displaying information on your CPU—its model name, speed, and so on.

Knowledge of these directories and their purposes is invaluable in properly administering a Linux system. For instance, understanding the purpose of directories like /bin, /sbin, /usr/bin, /usr/local/bin, and others will help you when it comes time to install a new program. Placing a program in the wrong location can cause problems at a later date. For example, if you put a binary file in /bin when it should go in /usr/local/bin, that program may later be overwritten or deleted during a system upgrade when leaving it intact would have been more appropriate.

Tools for Locating Files

You use file-location commands to locate a file on your computer. Most frequently, these commands help you locate a file by name, but sometimes you can use other criteria, such as modification date. These commands can search a directory tree (including root, which scans the entire system) for a file matching the specified criteria in any subdirectory.

The *find* Command

The find utility implements a brute-force approach to finding files. This program finds files by searching through the specified directory tree, checking filenames, file creation dates, and so on to locate the files that match the specified criteria. Because of this method of operation, find tends to be slow, but it's very flexible and is very likely to succeed, assuming the file for which you're searching exists. The find syntax is as follows:

```
find [path...] [expression...]
```

You can specify one or more paths in which find should operate; the program will restrict its operations to these paths. The *expression* is a way of specifying what you want to find. The man page for find includes information on these expressions, but some of the more common enable you to search by various common criteria:

Search by filename   You can search for a filename using the -name *pattern* expression. Doing so finds files that match the specified *pattern*. If *pattern* is an ordinary filename, find matches that name exactly. You can use wildcards if you enclose *pattern* in quotes, and find will locate files that match the wildcard filename.

Search by permission mode   If you need to find files that have certain permissions, you can do so by using the -perm *mode* expression. The *mode* may be expressed either symbolically or in octal form. If you precede *mode* with a +, find locates files in which *any* of the specified permission bits are set. If you precede *mode* with a -, find locates files in which *all* the specified permission bits are set.

Search by file size   You can search for a file of a given size with the -size *n* expression. Normally, *n* is specified in 512-byte blocks, but you can modify this by trailing the value with a letter code, such as c for bytes or k for kilobytes.

Search by group   The -gid *GID* expression searches for files whose group ID (GID) is set to *GID*. The -group *name* option locates files whose group name is *name*. The former can be handy if the GID has been orphaned and has no name, but the latter is generally easier to use.

Search by user ID   The -uid *UID* expression searches for files owned by the user whose user ID (UID) is *UID*. The -user *name* option searches for files owned by *name*. The former can be handy if the UID has been orphaned and has no name, but the latter is generally easier to use.

Restrict search depth   If you want to search a directory and, perhaps, some limited number of subdirectories, you can use the -maxdepth *levels* expression to limit the search.

There are many variant and additional options; find is a very powerful command. As an example of its use, consider the task of finding all C source code files, which normally have names that end in .c, in all users' home directories. If these home directories reside in /home, you might issue the following command:

```
# find /home -name "*.c"
```

The result will be a listing of all the files that match the search criteria.

> Ordinary users may use find, but it doesn't overcome Linux's file permission features. If you lack permission to list a directory's contents, find will return that directory name and the error message Permission denied.

The *locate* Command

The locate utility works much like find if you want to find a file by name, but it differs in two important ways:

- The locate tool is far less sophisticated in its search options. You normally use it to search only on filenames, and the program returns all files that contain the specified string. For instance, when searching for rpm, locate will return other programs, like gnorpm and rpm2cpio.

- The locate program works from a database that it maintains. Most distributions include a cron job that calls locate with options that cause it to update its database periodically, such as once a night or once a week. (You can also use the updatedb command to do this task at any time.) For this reason, locate may not find recent files, or it may return the names of files that no longer exist. If the database update utilities omit certain directories, files in them won't be returned by a locate query.

Because `locate` works from a database, it's typically much faster than `find`, particularly on system-wide searches. It's likely to return many false alarms, though, especially if you want to find a file with a short name. To use it, type **locate *search-string***, where *search-string* is the string that appears in the filename.

> Some Linux distributions use `slocate` rather than `locate`. The `slocate` program includes security features to prevent users from seeing the names of files in directories they should not be able to access. On most systems that use `slocate`, the `locate` command is a link to `slocate`, so `locate` implements `slocate`'s security features. A few distributions, including SuSE, don't install either `locate` or `slocate` by default.

The *whereis* Command

The `whereis` program searches for files in a restricted set of locations, such as standard binary file directories, library directories, and `man` page directories. This tool does *not* search user directories or many other locations that are easily searched by `find` or `locate`. The `whereis` utility is a quick way to find program executables and related files like documentation or configuration files.

The `whereis` program returns filenames that begin with whatever you type as a search criterion, even if those files contain extensions. This feature often turns up configuration files in `/etc`, `man` pages, and similar files. To use the program, type the name of the program you want to locate. For instance, the following command locates `ls`:

```
$ whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.bz2
```

The result shows both the `ls` executable (`/bin/ls`) and the `ls` man page. The `whereis` program accepts several parameters that modify its behavior in various ways. These are detailed in the program's `man` page.

The *which* Command

Considered as a search command, `which` is very weak; it merely searches your path for the command that you type and lists the complete path to the first match it finds. (You can search for all matches by adding the `-a` option, though.) For instance, you might want to know where the `xterm` program is located:

```
$ which xterm
/usr/bin/xterm
```

Because the files that `which` finds are on your path, it won't help you to run these programs. Instead, it's likely to be useful if you need to know the complete path for some reason—say, because you want to call the program from a script and don't want to make assumptions about the path available to the script and so want to include the complete path in the script.

EXERCISE 4.2

Locating Files

This exercise demonstrates several methods of locating files. You'll locate the `startx` program. (If your system doesn't have X installed, you can try searching for another program or file, such as `pwd` or `fstab`. You may need to change the path passed to `find` in step 5, though.) To find a file, follow these steps:

1. Log into the Linux system as a normal user.

2. Launch an `xterm` from the desktop environment's menu system if you used a GUI login method.

3. Type **`locate startx`**. The system should display several filenames that include the string `startx`. This search should take very little time. (A few distributions lack the `locate` command, so this step won't work on some systems.)

4. Type **`whereis startx`**. The system responds with the names of a few files that contain the string `startx`. Note that this list is slightly different than the list returned by step 3 but that the search proceeded quite quickly.

5. Type **`find /usr -name startx`**. This search will take longer, and when run as an ordinary user, it will most likely return several `Permission denied` error messages. It should also return a single line listing the `/usr/X11R6/bin/startx` program file. Note that this command searched only `/usr`. If you had searched `/usr/X11R6`, the command would have taken less time; if you'd searched `/`, the command would have taken more time.

6. Type **`which startx`**. This search will complete almost instantaneously, returning the complete filename of the first instance of `startx` the system finds on its path.

Summary

File and filesystem management are basic to being able to administer or use a Linux system. The most basic of these basic tasks are filesystem tasks—the ability to mount filesystems, check their health, and repair ailing filesystems. Once a filesystem is mounted, you may want to periodically check to see how full it is lest you run out of disk space. Various commands are useful to both users and administrators for copying, moving, renaming, and otherwise manipulating files and directories. You may also want to set up access controls, both to limit the amount of disk space users may consume and to limit who may access specific files and directories. Finally, Linux provides tools to help you locate files using various criteria.

Exam Essentials

Describe commands that help you monitor disk use.   The `df` command provides a one-line summary of each mounted filesystem's size, available space, free space, and percentage of space used. The `du` command adds up the disk space used by all the files in a specified directory tree and presents a summary by directory and subdirectory.

Summarize the tools that can help keep a filesystem healthy.   The `fsck` program is a front-end to filesystem-specific tools such as `e2fsck` and `fsck.xfs`. By whatever name, these programs examine a filesystem's major data structures for internal consistency and can correct minor errors.

Explain how filesystems are mounted in Linux.   The `mount` command ties a filesystem to a Linux directory; once the filesystem is mounted, its files can be accessed as part of the mount directory. The `/etc/fstab` file describes permanent mappings of filesystems to mount points; when the system boots, it automatically mounts the described filesystems unless they use the `noauto` option (which is common for removable disks).

Describe commands used to copy, move, and rename files in Linux.   The `cp` command copies files, as in **cp first second** to create a copy of `first` called `second`. The `mv` command does double duty as a file-moving and a file-renaming command. It works much like `cp`, but `mv` moves or renames the file rather than copying it.

Summarize Linux's directory-manipulation commands.   The `mkdir` command creates a new directory, while `rmdir` deletes a directory. You can also use many file-manipulation commands, such as `mv` and `rm` (with its `-r` option) on directories.

Explain the difference between hard and symbolic links.   Hard links are duplicate directory entries that both point to the same inode and hence to the same file. Symbolic links are special files that point to another file or directory by name. Hard links must reside on a single filesystem, but symbolic links may point across filesystems.

Describe Linux's file ownership system.   Every file has an owner and a group, identified by number. File permissions can be assigned independently to the file's owner, the file's group, and all other users.

Explain Linux's file permissions system.   Linux provides independent read, write, and execute permissions for the file's owner, the file's group, and all other users, resulting in nine main permission bits. Special permission bits are also available, enabling you to launch program files with modified account features or alter the rules Linux uses to control who may delete files.

Summarize the commands Linux uses to modify permissions.   The `chmod` command is Linux's main tool for setting permissions. You can specify permissions using either an octal (base 8) mode or a symbolic notation. The `chown` and `chgrp` commands enable you to change the file's owner and group, respectively. (The `chown` command can do both but can only be run by `root`.)

Describe the prerequisites of using Linux's disk quota system.   Linux's disk quota system requires support in the Linux kernel for the filesystem on which quotas are to be used. You must also run the `quotaon` command, typically from a SysV startup script, to enable this feature.

Explain how quotas are set. You can edit quotas for an individual user via the `edquota` command, as in **`edquota larry`** to edit `larry`'s quotas. This command opens an editor on a text file that describes the user's quotas. You can change this description, save the file, and exit from the editor to change the user's quotas.

Summarize how Linux's standard directories are structured. Linux's directory tree begins with the root (`/`) directory, which holds mostly other directories. Specific directories may hold specific types of information, such as user files in `/home` and configuration files in `/etc`. Some of these subdirectories and their subdirectories may in fact be separate partitions, which helps isolate data in the event of filesystem corruption.

Describe the major file-location commands in Linux. The `find` command locates files by brute force, searching through the directory tree for files that match the criteria you specify. The `locate` (or `slocate`) command searches a database of files in important directories. The `whereis` command searches a handful of important directories, while `which` searches the path.

Review Questions

1. Which of the following options is used with `fsck` to force it to use a particular filesystem type?

    A. `-A`

    B. `-N`

    C. `-t`

    D. `-C`

2. Which of the following pieces of information can `df` *not* report?

    A. How long the filesystem has been mounted

    B. The number of inodes used on an ext3fs partition

    C. The filesystem type of a partition

    D. The percentage of available disk space used on a partition

3. What is an advantage of a journaling filesystem over a conventional (non-journaling) filesystem?

    A. Journaling filesystems are older and better tested than non-journaling filesystems.

    B. Journaling filesystems never need to have their filesystems checked with `fsck`.

    C. Journaling filesystems support Linux ownership and permissions; non-journaling filesystems don't.

    D. Journaling filesystems require shorter disk checks after a power failure or system crash.

4. You want to use `/mnt/cdrom` to access your CD-ROM drive, so you insert a CD-ROM into the drive and type **mount /mnt/cdrom /dev/cdrom** as `root`, but you receive the error message `/mnt/cdrom is not a block device`. Why did this happen?

    A. You must first prepare the mount point by typing **mountpoint /mnt/cdrom**; only then will the `mount` command succeed.

    B. The command reverses the order of the CD-ROM device file and the mount point; it should be **mount /dev/cdrom /mnt/cdrom**.

    C. The `/dev/cdrom` filename is not valid; you must determine what device file is associated with your CD-ROM drive.

    D. The CD-ROM is defective or the CD-ROM drive is malfunctioning. Try another CD-ROM and, if necessary, replace the drive.

5. What is wrong with the following `/etc/fstab` file entry? (Select all that apply.)

`/dev/hda8  nfs  default  0 0`

    A. The entry is missing a mount point specification.

    B. All `/etc/fstab` fields should be separated by commas.

    C. The `default` option may only be used with ext2 filesystems.

    D. `/dev/hda8` is a disk partition, but `nfs` indicates a network filesystem.

6. You want to discover the sizes of several dot files in a directory. Which of the following commands might you use to do this?

    A. `ls -la`

    B. `ls -p`

    C. `ls -R`

    D. `ls -d`

7. You want to move a file from your hard disk to a floppy disk. Which of the following is true?

    A. You'll have to use the `--preserve` option to `mv` to keep ownership and permissions set correctly.

    B. The `mv` command will adjust filesystem pointers without physically rewriting data if the floppy uses the same filesystem type as the hard disk partition.

    C. You must use the same filesystem type on both media to preserve ownership and permissions.

    D. The `mv` command will delete the file on the hard disk after copying it to the floppy.

8. You type **`mkdir one/two/three`** and receive an error message that reads, in part, `No such file or directory`. What can you do to overcome this problem? (Select all that apply.)

    A. Add the `--parents` parameter to the `mkdir` command.

    B. Issue three separate `mkdir` commands: **`mkdir one`**, then **`mkdir one/two`**, then **`mkdir one/two/three`**.

    C. Type **`touch /bin/mkdir`** to be sure the `mkdir` program file exists.

    D. Type **`rmdir one`** to clear away the interfering base of the desired new directory tree.

9. You want to create a link from your home directory on your hard disk to a directory on a CD-ROM drive. Which of the following types of links might you use?

    A. Only a symbolic link.

    B. Only a hard link.

    C. Either a symbolic or a hard link.

    D. None of the above; such links are not possible under Linux.

10. What command would you type to change the ownership of `somefile.txt` from `ralph` to `tony`?

    A. `chown ralph:tony somefile.txt`

    B. `chmod somefile.txt tony`

    C. `chown somefile.txt tony`

    D. `chown tony somefile.txt`

11. Typing `ls -ld wonderjaye` reveals a symbolic file mode of `drwxr-xr-x`. Which of the following is true? (Select all that apply.)

   A. `wonderjaye` is a symbolic link.

   B. `wonderjaye` is an executable program.

   C. `wonderjaye` is a directory.

   D. `wonderjaye` may be read by all users of the system.

12. When should programs be configured SUID `root`?

   A. At all times; this permission is required for executable programs.

   B. Whenever a program should be able to access a device file.

   C. Only when they require `root` privileges to do their job.

   D. Never; this permission is a severe security risk.

13. Which of the following commands would you type to give all users read access to the file `myfile.txt`? (Assume that you're the owner of `myfile.txt`.)

   A. `chmod 741 myfile.txt`

   B. `chmod 0640 myfile.txt`

   C. `chmod u+r myfile.txt`

   D. `chmod o+r myfile.txt`

14. Which of the following umask values will result in files with `rw-r-----` permissions?

   A. 640

   B. 210

   C. 022

   D. 027

15. You see the `usrquota` and `grpquota` options in the `/etc/fstab` entry for a filesystem. What is the consequence of these entries?

   A. Quota support will be available if it's compiled into the kernel; it will be automatically activated when you mount the filesystem.

   B. Quota support will be available if it's compiled into your kernel, but you must activate it with the `quotaon` command.

   C. Quota support will be disabled on the filesystem in question unless you activate it with the `quotaon` command.

   D. Nothing; these options are malformed and so will have no effect.

16. Which of the following commands can be used to summarize the quota information on all filesystems?

    A. `repquota`

    B. `repquota -a`

    C. `quotacheck`

    D. `quotacheck -a`

17. You've installed a commercial spreadsheet program, called WonderCalc, on a workstation. In which of the following directories are you *most* likely to find the program executable file?

    A. `/usr/sbin`

    B. `/etc/X11`

    C. `/bin`

    D. `/opt/wcalc/bin`

18. Which of the following file-location commands is likely to take the *most* time to find a file that might be located anywhere on the computer?

    A. The `find` command.

    B. The `locate` command.

    C. The `whereis` command.

    D. They're all equal in speed.

19. You want to track down all the files in /home that are owned by `karen`. Which of the following commands will do the job?

    A. `find /home -uid karen`

    B. `find /home -user karen`

    C. `locate /home -username karen`

    D. `locate /home karen`

20. What can you conclude from the following interaction?

```
$ which man
/usr/bin/man
```

    A. The only file called `man` on the system is in `/usr/bin`.

    B. The `/usr/bin/man` program was installed by system package tools.

    C. The `/usr/bin/man` program will be run by any user who types **man**.

    D. The first instance of the `man` program, in path search order, is in `/usr/bin`.

Answers to Review Questions

1. C. The –t option is used to tell fsck what filesystem to use. Normally, fsck determines the filesystem type automatically. The –A option causes fsck to check all the filesystems marked to be checked in /etc/fstab. The –N option tells fsck to take no action and to display what it would normally do without actually doing it. The –C option displays a text-mode progress indicator of the check process.

2. A. A default use of df reports the percentage of disk space used. The number of inodes and filesystem types can both be obtained by passing parameters to df. This utility does *not* report how long a filesystem has been mounted.

3. D. The journal of a journaling filesystem records pending operations, resulting in quicker disk checks after an uncontrolled shutdown. Contrary to option A, journaling filesystems are, as a class, newer than non-journaling filesystems; in fact, the journaling ext3fs is built upon the non-journaling ext2fs. Although disk checks are quicker with journaling filesystems than with non-journaling filesystems, non-journaling filesystems do have fsck utilities, and these may still need to be run from time to time. All Linux native filesystems support Linux ownership and permissions; this isn't an advantage of journaling filesystems, contrary to option C.

4. B. The mount command takes a device filename first and a mount point second, so option B is correct. There is no need for special mount point preparation, as option A implies, other than the existence of the mount point directory. Contrary to option C, /dev/cdrom is often a valid device file for CD-ROMs (or more precisely, a symbolic link to a valid device file). Furthermore, the error message mentions /mnt/cdrom, not /dev/cdrom. Defective CD-ROMs and drives can certainly cause problems, as option D implies, but this error message refers to block devices, suggesting the problem is one of Linux configuration, not media viability.

5. A, D. A mount directory must be specified between the device entry (/dev/hda8) and the filesystem type code (nfs). The nfs filesystem type code may only be used with an NFS export specification of the form *server:/export* as the device specification. Fields in /etc/fstab are separated by spaces or tabs, not commas (but commas are used between individual options if several options are specified in the options column). The default option may be used with *any* filesystem type.

6. A. The -1 parameter produces a long listing, including file sizes. The -a parameter produces a listing of all files in a directory, including the dot files. Combining the two produces the desired information (along with information on other files). The -p, -R, and -d options do not have the specified effects.

7. D. When moving from one partition or disk to another, mv must necessarily read and copy the file and then delete the original if that copy was successful. If both filesystems support ownership and permissions, they'll be preserved; mv doesn't need an explicit --preserve option to do this, and this preservation does not rely on having exactly the same filesystem types. Although mv doesn't physically rewrite data when moving within a single low-level filesystem, this approach cannot work when you are copying to a separate low-level filesystem (such as from a hard disk to a floppy disk); if the data isn't written to the new location, it won't be accessible should the disk be inserted in another computer.

8. A, B. If you try to create a directory inside a directory that doesn't exist, mkdir responds with a No such file or directory error. The --parents parameter tells mkdir to automatically create all necessary parent directories in such situations. You can also manually do this by creating each necessary directory separately. (It's possible that mkdir one wouldn't be necessary in this example if the directory one already exists. No harm will come from trying to create a directory that already exists, although mkdir will return a File exists error.)

9. A. Symbolic links can point across filesystems, so creating a symbolic link from one filesystem (in which your home directory resides) to another (on the CD-ROM) isn't a problem. Hard links, as in options B and C, are restricted to a single filesystem and so won't work for the described purpose. Because symbolic links will work as described, option D is incorrect.

10. D. Option D is the correct command. Typing **chown ralph:tony somefile.txt**, as in option A, sets the owner of the file to ralph and the group to tony. The chmod command used in option B is used to change file permissions, not ownership. Option C reverses the order of the filename and the owner.

11. C, D. The d character that leads the mode indicates that the file is actually a directory, while the r symbol in the r-x triplet at the end of the symbolic mode indicates that all users of the system have read access to the directory. Symbolic links are denoted by leading l characters, which this mode lacks, so option A is incorrect. Although the x symbols usually denote executable program files, as specified in option B, in the case of directories this permission bit indicates that the directory's contents may be searched; executing a directory is meaningless.

12. C. The set user ID (SUID) bit enables programs to run as the program's owner rather than as the user who ran them. This makes SUID root programs risky, so setting the SUID bit on root-owned programs should be done only when it's required for the program's normal functioning, as stated in option C. This should certainly *not* be done for all programs because the SUID bit is *not* required of all executable programs as option A asserts. Although the SUID root configuration does enable programs to access device files, the device files' permissions can be modified to give programs access to those files, if this is required, so option B is incorrect. Although SUID root programs are a security risk, as stated in option D, they're a necessary risk for a few programs, so option D goes too far.

13. D. Using symbolic modes, the o+r option adds read (r) permissions to the world (o). Thus, option D is correct. Option A sets the mode to rwxr----x, which is a bit odd and does not provide world read access to the file, although it does provide world execute access. Option B sets the mode to rw-r-----, which gives the world no access whatsoever to the file. Option C adds read access to the file for the owner (u) if the owner does not already have this access; it doesn't affect the world permissions.

14. D. Option D, 027, removes write permissions for the group and all world permissions. (Files normally don't have execute permissions set, but explicitly removing write permissions when removing read permissions ensures reasonable behavior for directories.) Option A, 640, is the octal equivalent of the desired rw-r----- permissions, but the umask sets the bits that are to be *removed* from permissions, not those that are to be set. Option B, 210, would remove write permission for the owner, but it would not remove write permission for the group, which is incorrect. This would also leave all world permissions open. Finally, option C, 022, would not remove world read permission.

15. B. Using quotas requires kernel support, the `usrquota` or `grpquota` (for user or group quotas) filesystem mount option, and activation via the `quotaon` command (which often appears in SysV startup scripts). Thus, option B is correct. Option A suggests that `quotaon` is not necessary, and option C suggests that `quotaon` is sufficient, but neither is true. The `usrquota` and `grpquota` options are both valid, so option D is incorrect.

16. B. The `repquota` utility is used to summarize the quota information on the filesystem. When used with the `-a` option, it will show this information for all filesystems. The `quotacheck` utility checks quota information on a disk and writes corrections.

17. D. The `/opt` directory tree exists to hold programs that aren't a standard part of a Linux distribution, such as commercial programs. These programs should install in their own directories under `/opt`; these directories usually have `bin` subdirectories of their own, although this isn't required. The `/usr/sbin` directory holds programs that are normally run only by the system administrator, and `/bin` holds critical basic binary files. The `/etc/X11` directory holds X-related configuration files. None of these directories is an appropriate place for a spreadsheet program.

18. A. The `find` utility operates by searching all files in a directory tree, and so it is likely to take a long time to search all a computer's directories. The `locate` program uses a precompiled database, and `whereis` searches a limited set of directories, so these commands will take less time.

19. B. The `find` command includes the ability to search by username using the `-user name` option, where *name* is the username; thus, option B is correct. The `-uid` option to `find` can also locate files owned by a user, but it takes a numeric user ID (UID) number as an argument, so option A isn't quite correct. The `locate` command provides no ability to search by user, so options C and D are incorrect.

20. D. The `which` program searches the path just as `bash` does, but it prints the path to the first executable program it finds on the path. Thus, option D is correct. The `which` program does not conduct an exhaustive search of the system, so there could be many more files called `man` on the system, contrary to option A. System package tools and `which` are not closely related; option B is incorrect. Although `/usr/bin/man` would be run when the user whose `which` output matches that in the question types **man**, this might not be true of others because the path can vary from one user to another. Thus, option C is incorrect.

The X Window System

THE FOLLOWING LINUX PROFESSIONAL INSTITUTE OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **1.110.1 Install & Configure XFree86 (weight: 5)**
- ✓ **1.110.2 Setup a display manager (weight: 3)**
- ✓ **1.110.4 Install & Customize a Window Manager Environment (weight: 5)**

Major modern desktop OSs all provide some form of *graphical user environment (GUI)*, which provides the windows, menus, dialog boxes, flexible fonts, and so on with which you're probably already familiar. In Linux, the main GUI is known as the *X Window System* (or *X* for short). X configuration is either very easy or moderately hard; most distributions today provide auto-detection and easy configuration options during installation, and these usually work correctly. When they don't, or when you want to tweak the configuration, you must delve into the X configuration file or use a GUI X configuration tool. Doing either requires that you know how X treats the video hardware, among other things.

Beyond basic X configuration are a few extra topics. These include fonts, GUI login tools, user desktop environments, and using X for remote access. Each of these topics is closely associated with basic X configuration, but they all go beyond it in one way or another, extending X's capabilities or providing more features for users, as described in this chapter.

Configuring Basic X Features

Basic X configuration sets features such as the mouse used, the keyboard layout, the screen resolution, the video refresh rate, the display color depth, and the video card you're using. Some of these options require telling X about what hardware you have installed, whereas others enable you to adjust settings on your hardware. In any event, before you proceed with actual configuration, you should know something about the X servers that are available for Linux because your selection will determine what additional tools are available and what files you may need to adjust manually. GUI and text-mode configuration utilities can help you configure X, but sometimes you must delve into the configuration files, so knowing their format is important. This requires that you know what the major option groups do so that you can adjust them.

X Server Options for Linux

Although X is by far the dominant GUI for Linux, several implementations of X are available:

XFree86   The dominant X server in Linux until 2004 was XFree86 (`http://www.xfree86.org`). This open-source server supports a wide array of video cards and input devices, and most Linux software is designed with XFree86 in mind. As I write, the most recent version is 4.5.0. Significant changes occurred between 3.3.6 and the 4.*x* series, and many utilities (including the `xf86config` and `XF86Setup` tools mentioned in the LPI objectives) work only with the old 3.3.6 and earlier versions of XFree86. Although a tiny number of systems must run XFree86 3.3.6 or earlier for driver support reasons, most systems today run XFree86 4.*x* or X.org-X11.

X.org-X11   In 2004, most Linux distributions shifted from XFree86 to X.org-X11 because of some licensing changes to XFree86. X.org-X11 6.7.0 was based on XFree86 4.3.99, but it's developed independently up to the current version (6.7.2 as I write). Because X.org-X11 is based on XFree86, the two are virtually identical in most important respects. One important difference is the name of the configuration file; another is the default location for fonts. Subsequent sections of this chapter point out these differences. Other differences are likely to emerge in the future as XFree86 and X.org-X11 develop independently. You can learn more at `http://www.x.org`.

Accelerated-X   The commercial Accelerated-X server from Xi Graphics (`http://www.xig.com`) is an alternative to the open-source XFree86 and X.org-X11. In practice, running Accelerated-X is seldom necessary, but if you have problems getting your video card working, you may want to look into Accelerated-X; its driver base is independent of the more popular open-source choices, so it's possible you'll have better luck with it. The Accelerated-X configuration tools and files are completely different from those described in "Methods of Configuring X" and "X Configuration Options," though, so you'll need to consult its own documentation for help. The rest of this chapter's topics still apply to Accelerated-X.

In practice, it's usually easiest to stick with whatever X server your distribution provides. For modern distributions, this is most often X.org-X11, but it might be XFree86, particularly if you're running an older distribution. For a handful of rare video cards, you may need to run the rather elderly XFree86 3.3.6 rather than a more recent version.

Methods of Configuring X

Configuring X has traditionally been a difficult process because the X configuration file includes many arcane options. The task is made simpler if you can use a configuration utility, and in fact most Linux distributions now run such a utility as part of their installation process. If the configuration utility doesn't do everything you want it to do, though, you may need to delve into the X configuration file to set options manually, so knowing something about its format will help a lot. You must also know how to go about restarting X in order to test your changes.

> The upcoming section "X Configuration Options" describes the major X features and how to control them in more detail.

X Configuration Utilities

X configuration utilities vary from one version of X to another. Most importantly, the tools for XFree86 3.3.6 and earlier are very different from those for XFree86 4.*x* and X.org-X11. The LPI exam still covers the XFree86 3.3.6 tools, but in the real world you're more likely to encounter the more recent tools.

Real World Scenario

Using Manufacturer-Provided Video Drivers

One of X's functions is to provide drivers that control the video card. XFree86, X.org-X11, and Accelerated-X all ship with a wide variety of drivers that support most video cards. Some cards, though (particularly very new ones), have weak support in the stock packages. XFree86 4.*x* and X.org-X11 both support a modular driver architecture, which means you can drop in a driver module for your card and, with minimal changes to your X configuration, use it without recompiling the main X package. Some video card manufacturers provide Linux video card drivers designed to work with XFree86 and X.org-X11. (Both X servers can use the same drivers.) Thus, if you have problems with the standard X video drivers, you may want to check with your video card manufacturer and the video card chipset manufacturers for Linux drivers.

Installing and using the manufacturer-provided video drivers is usually a matter of extracting files from a tarball and making a few configuration file changes. Consult the documentation that comes with these drivers for details. Many of these drivers are particularly helpful for enabling 3D acceleration features of modern cards. This support is most important for certain Linux games, but some non-game software can also benefit from 3D acceleration.

Note that some manufacturers have a habit of placing the standard Linux drivers (or, in this case, the standard XFree86 or X.org-X11 drivers) on their website. Thus, you should try to ascertain whether the drivers are genuinely different from what you already have before you bother installing them.

One problem with manufacturer-supplied drivers is that they're often proprietary. You might not have source code, which means that the drivers might not work on more exotic CPUs, and the drivers could cease working with a future upgrade to your X server.

Configuration Tools for XFree86 3.3.6 and Earlier

Many configuration tools are available for XFree86 3.3.6 and before. Most distributions ship with at least one of the following tools, and sometimes more:

xf86config This is the crudest of the XFree86 3.3.6 configuration programs. It runs entirely in text mode, and it asks a series of questions regarding your hardware and configuration preferences. The program asks about your mouse type (including brand and device file), keyboard type (including number of keys and country), monitor specifications (horizontal and vertical refresh rates), and video card (chipset or card manufacturer, amount of video RAM, and default resolution). Because of its simple user interface, it can be used to create an initial X configuration. The xf86config utility offers no provision for correcting errors, though, so if you mistype an entry, you must stop it and start again from scratch.

Xconfigurator This program runs in text mode when launched from a text-based login. In some distributions, it runs in GUI mode when launched from X. In either case, it uses menus to help you pick appropriate settings, but it still steps you through the configuration in a fixed

order. As a general rule, it's easier to use than xf86config. Recent versions support both XFree86 3.3.6 and 4.*x*.

XF86Setup This program runs only in GUI mode, as shown in Figure 5.1, so it works only once X is running at least minimally. This program can be used to reconfigure X from a working but non-optimal mode. It enables you to move around the configuration as you see fit by selecting any of the major systems from the buttons along the top of the window (Mouse, Keyboard, Card, and so on).

As a general rule, Xconfigurator can be a good tool to use for configuring a new system for the first time, and XF86Setup is good for adjusting an existing system without digging into XFree86's XF86Config file. (This file has the same name as one configuration utility, but its case is different.) Unfortunately for those forced to use XFree86 3.3.6, these tools are very hard to find and use with modern distributions, so you may be forced to use xf86config or edit the configuration file manually. Fortunately, most people today can use XFree86 4.*x* or X.org-X11 and the native configuration tools for these X servers.

Sometimes one configuration tool will produce a working X system but another won't. This is particularly likely when you're using finicky hardware, such as the LCD monitors used on notebook computers. If you can't seem to get a working X configuration out of one utility program, try another.

F I G U R E   5 . 1   XF86Setup enables you to select which options you want to configure in each of several major categories.
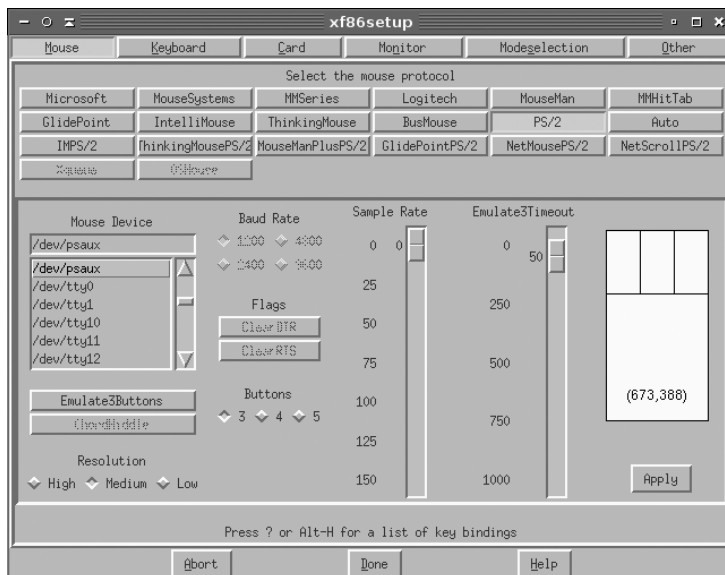
Configuration Tools for XFree86 4.*x* and X.org-X11

Configuration tools for XFree86 4.*x* and X.org-X11 are identical but are (with one notable exception) different from those for XFree86 3.3.6 and earlier:

The X server itself   The X server itself includes the capacity to query the hardware and produce a configuration file. To do so, type **XFree86 -configure** (for XFree86) or **Xorg -configure** (for X.org-X11) when no X server is running. The result should be a file called /root/XF86Config.new (for XFree86) or /root/xorg.conf.new (for X.org-X11). This file might not produce optimal results, but it is at least a starting point for manual modifications.

Xconfigurator   Current versions of Xconfigurator can produce and modify the XFree86 4.*x* XF86Config file format as well as the 3.3.*x* format of the file. Red Hat (Xconfigurator's developer) has abandoned this tool in favor of GUI utilities, though.

Distribution-specific tools   Many modern distributions ship with their own custom X configuration tools. These include Red Hat's (and Fedora's) Display Settings tool (accessible from the default desktop menu or by typing **system-config-xfree86** in an xterm) and SuSE's YaST and YaST2. These tools frequently resemble the distribution's install-time X configuration tools, which can vary substantially.

xf86cfg or xorgcfg   This utility works only on XFree86 4.*x* or X.org-X11 (each package ships with a version named after itself). This program is another that works only once X is already running. Its user interface (shown in Figure 5.2), like that of XF86Setup, enables you to jump around to configure different elements in whatever order you like. In xf86cfg/xorgcfg, you right-click an icon and select the Configure option to configure the element, or you can select other options (Remove, Disable, and so on) to perform other actions.

F I G U R E   5 . 2   The xf86cfg/xorgcfg program lets you configure XFree86 4.x or X.org-X11 using point-and-click operations.
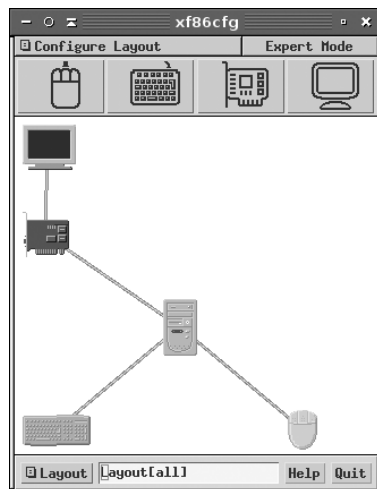
All of these utilities gather the same type of information needed by XFree86 3.3.6 configuration tools, which in turn is the same type of information needed to manually configure X. Your best bet for understanding these tools and what they want is to understand the underlying X configuration file's format and contents.

The X Configuration File Format

The X configuration file's name and location varies with the version of X being run:

XFree86 3.3.6 and earlier   The X configuration file's name is `XF86Config`, and the file is most commonly located in `/etc/X11` or `/etc`.

XFree86 4.*x*   The XFree86 4.*x* configuration file format is slightly different from that of XFree86 3.3.6 and earlier, so to support the transition period, XFree86 4.*x* supports a configuration file called `XF86Config-4`, which is typically located in `/etc/X11`. If this file isn't found, XFree86 4.*x* looks for a file called `XF86Config` in `/etc/X11` or `/etc`.

X.org-X11   This server's configuration file is called `xorg.conf`, and it's usually stored in `/etc/X11`, although `/etc` and several other locations are also acceptable to the server. This file's format is exactly the same as for the XFree86 4.*x* configuration file.

All three of these classes of X server use configuration files that are broken down into multi-line sections, one section for each major feature. These sections begin with a line consisting of the keyword `Section` and the section name in quotes and end with the keyword `EndSection`:

```
Section "InputDevice"
    Identifier  "Keyboard0"
    Driver      "kbd"
    Option      "XkbModel" "pc105"
    Option      "XkbLayout" "us"
    Option      "AutoRepeat" "500 200"
EndSection
```

This section tells X about the keyboard—its model, layout, and so on. Details for the sections you're most likely to need to adjust are described shortly, in "X Configuration Options."

For the most part, the different X servers support the same sections and most of the same option names. A few exceptions to this rule do exist, though:

▪ The `Option` keyword is not used in XFree86 3.3.6 and earlier. Instead, the option name (such as `XkbLayout` or `AutoRepeat` in the preceding example) appears without quotes as the first word on the line.

▪ XFree86 3.3.6 and earlier don't use the `ServerLayout` section, described later in "Putting It All Together."

▪ XFree86 3.3.6 and earlier lack the `Identifier` and `Driver` lines, which are common in the XFree86 4.*x* and X.org-X11 configuration files.

▪ Some section-specific features vary between versions. I describe the most important of these in the coming pages.

The X Configure-and-Test Cycle

If your X configuration isn't working correctly, you need to be able to modify that configuration and then test it. Many Linux distributions configure the system to start X automatically; however, starting X automatically can make it difficult to test the X configuration. To a new Linux administrator, the only obvious way to test a new configuration is to reboot the computer.

A better solution is to kick the system into a mode in which X is *not* started automatically. On most distributions, this goal can be achieved by typing **telinit 3**. This action sets the computer to use runlevel 3, in which X normally doesn't run. Chapter 6, "The Boot Process and Scripts," covers runlevels in more detail, but for now, know that setting the system to a runlevel of 3 normally shuts down the X session that launched automatically at system startup.

> **NOTE** Some distributions, such as Debian and Gentoo, don't use runlevels as a signal for whether or not to start X. With such distributions, you must shut down the GUI login server by typing **/etc/init.d/xdm stop**. (You may need to change xdm to gdm or kdm, depending on your configuration.)

Once the X session is shut down, you can log in using a text-mode login prompt and tweak your X settings manually, or you can use text-based X configuration programs. You can then type **startx** to start the X server again. If you get the desired results, quit from X and type **telinit 5** (**/etc/init.d/xdm start** in Debian and other distributions that don't use runlevels to start the GUI login prompt) to restore the system to its normal X login screen. If after typing **startx** you don't get the results you want, you can end your X session and try modifying the system some more.

If X is working minimally but you want to modify it using X-based configuration tools, you can do so after typing **startx** to get a normal X session running. Alternatively, you can reconfigure the system before taking it out of the X-enabled runlevel.

Another approach to restarting X is to leave the system in its X-enabled runlevel and then kill the X server. The Ctrl+Alt+Backspace keystroke does this on many systems, or you can do it manually with the `kill` command after finding the appropriate process ID with the `ps` command, as shown here:

```
# ps ax | grep X
1375 ?   S  6:32 /etc/X11/X –auth /etc/X11/xdm/authdir/
# kill 1375
```

This approach works better on systems that don't map the running of X to specific runlevels, such as Debian and its derivatives.

X Configuration Options

When editing the X configuration file, the best approach is usually to identify the feature that's not working and zero in on the section that controls this feature. You can then edit that section, save your changes, and test the new configuration. In XFree86 4.*x* and X.org-X11, the major

sections described here are called `Module`, `InputDevice`, `Monitor`, `Device`, `Screen`, and `ServerLayout`. You're likely to have two `InputDevice` sections, one for the keyboard and one for the mouse. (In XFree86 3.3.6 and earlier, the mouse is handled by a separate `Pointer` section.) The section order doesn't matter.

> **NOTE** Fonts are a complex enough topic that they're described in more detail a bit later, in "Configuring X Fonts." Part of this configuration is handled in the `Files` section.

Loading Modules

The `Module` section controls the loading of X server modules—drivers for specific features or hardware. A typical example looks like this:

```
Section "Module"
    Load  "dbe"
    Load  "extmod"
    Load  "fbdevhw"
    Load  "glx"
    Load  "record"
    Load  "freetype"
    Load  "type1"
    Load  "dri"
EndSection
```

Each module is named (`dbe`, `extmod`, and so on) and is loaded by name using the `Load` option. Most of these module names can be deciphered with a bit of knowledge about the features they control. For instance, `freetype` and `type1` handle TrueType and Adobe Type 1 font rendering, respectively. If you're perusing your `Module` section and see modules you don't understand, though, you shouldn't worry about it; generally speaking, modules that are configured automatically are necessary for normal operation, or at least they do no harm.

For the most part, if an X configuration works, you shouldn't try to adjust the `Module` section, even if you want to tweak the X configuration. Sometimes, though, you'll need to add lines to this section or occasionally remove them. This is particularly likely to be necessary if you're activating 3D acceleration support or some sort of exotic feature. In such cases, you should consult the documentation for the feature you want to activate.

Setting the Keyboard

The keyboard is one of two common input devices configured via an `InputDevice` section:

```
Section "InputDevice"
    Identifier  "Keyboard0"
    Driver      "kbd"
```

```
    Option        "XkbModel" "pc105"
    Option        "XkbLayout" "us"
    Option        "AutoRepeat" "500 200"
EndSection
```

The `Identifier` line provides a label that's used by another section (`ServerLayout`, described in "Putting It All Together"). This line and the `ServerLayout` section are not present in XFree86 3.3.6 or earlier. The string given on this line is arbitrary, but for a keyboard, a name such as this example's `Keyboard0` will help you understand the file.

The `Driver` line tells X what driver to use to access the keyboard. This should be `kbd` or `Keyboard`, depending on your X server. Unless your keyboard isn't working at all, you shouldn't adjust this line.

The `Option` lines set various options that adjust various keyboard features, such as the model, the layout, and the repeat rate. For the most part, the defaults work well; however, you might want to change the `AutoRepeat` option or add it if it's not present. This option tells X when to begin repeating characters when you hold down a key and how often to repeat them. It takes two numbers as values, enclosed in quotes: the time until the first repeat and the time between subsequent repeats, both times expressed in milliseconds (ms). In the preceding example, the system waits 500ms (half a second) for the first repeat and then 200ms for each subsequent repeat (that is, five repeats per second).

> **NOTE** Many desktop environments and other user-level utilities provide tools to set the keyboard repeat rate. Thus, the options you set in the X configuration file are used as defaults only and might or might not have any real effect for users.

Setting the Mouse

A second `InputDevice` section controls how X treats the mouse:

```
Section "InputDevice"
    Identifier  "Mouse0"
    Driver      "mouse"
    Option      "Protocol" "IMPS/2"
    Option      "Device" "/dev/input/mice"
    Option      "Emulate3Buttons" "no"
    Option      "ZAxisMapping" "4 5"
EndSection
```

> **NOTE** In XFree86 3.3.6 and earlier, this section is called `Pointer` rather than `InputDevice`.

As with the keyboard, the `Identifier` line is used in the `ServerLayout` section to tell X which input device to use. The `Driver` line identifies the driver to use: `mouse`. The `Option` lines set mouse control options. The most important of these are `Device` and `Protocol`.

The `Device` line tells X what Linux device file to read to access the mouse. In this example, it's `/dev/input/mice`, but other common possibilities include `/dev/mouse` (a pointer to the real mouse device, whatever its name), `/dev/psaux` (for the PS/2 mouse port), `/dev/usb/usbmouse` (an old identifier for USB mice), `/dev/ttyS0` (the first RS-232 serial port mouse), and `/dev/ttyS1` (the second RS-232 serial port mouse). If your mouse is working at all (even if its motions are erratic), don't change this line. If your mouse isn't working, you may need to experiment.

The `Protocol` option tells X what signals to expect from the mouse for various movements and button presses. The most common protocol for new mice today is `IMPS/2`, which stands for Microsoft's Intellimouse variant on the PS/2 mouse protocol. (Note that "PS/2" is both a hardware interface and a software protocol and many USB mice use the PS/2 mouse protocol even though they don't use the PS/2 mouse port.) If your mouse has a scroll wheel, chances are this is the protocol you should use. If your mouse is older, you may need to try older protocols, such as `PS/2`, `Microsoft`, or `Logitech`.

Additional options are usually less critical than the `Device` and `Protocol` options. The `Emulate3Buttons` option tells X whether or not to treat a chord (that is, a simultaneous press) of both buttons on a two-button mouse as if it were a middle button press. This option is usually disabled on three-button mice and scroll mice (the scroll wheel does double duty as a middle mouse button). The `ZAxisMapping` option in the preceding example maps the scroll wheel actions to the fourth and fifth buttons, because X must treat scroll wheels as if they were buttons. When you scroll up or down, these "button" presses are generated. Software can detect this and take appropriate actions.

Setting the Monitor

Some of the trickiest aspects of X configuration relate to the monitor options. You set these in the `Monitor` section, which has a tendency to be quite large, particularly in XFree86 3.3.6 and earlier. A shortened `Monitor` section looks like this:

```
Section "Monitor"
    Identifier "Monitor0"
    ModelName  "VisionMaster Pro 450"
    HorizSync  27.0-115.0
    VertRefresh 50.0-160.0
    # My custom 1360x1024 mode
    Modeline "1360x1024" 197.8 \
             1360 1370 1480 1752 \
             1024 1031 1046 1072 -HSync -VSync
EndSection
```

As in the keyboard and mouse configurations, the `Identifier` option is a free-form string that contains information that's used to identify a monitor. The `Identifier` can be just about anything you like. Likewise, the `ModelName` option also can be anything you like; it's used mainly for your own edification when reviewing the configuration file.

As you continue down the section, you'll see the `HorizSync` and `VertRefresh` lines, which are extremely critical; they define the range of horizontal and vertical refresh rates that the monitor can accept, in kilohertz (kHz) and hertz (Hz), respectively. Together, these values determine the maximum resolution and refresh rate of the monitor. Despite the name, the `HorizSync` item alone doesn't determine the maximum horizontal refresh rate. Rather, this value, the `VertRefresh` value, and the resolution determine the monitor's maximum refresh rate. X selects the maximum refresh rate that the monitor will support given the resolution you specify in other sections. Some X configuration utilities show a list of monitor models or resolution and refresh rate combinations (such as "800×600 at 72 Hz"). You select an option and the utility then computes the correct values based on that selection. This approach is often simpler to handle, but it's less precise than entering the exact horizontal and vertical sync values. You should enter these values from your monitor's manual.

> **WARNING** Don't set random horizontal and vertical refresh rates; particularly on older hardware, setting these values too high can actually damage a monitor. (Modern monitors ignore signals presented at too high a refresh rate.)

To settle on a resolution, X looks through a series of *mode lines,* which are specified via the `Modeline` option. Computing mode lines is tricky, so I don't recommend you try it unless you're skilled in such matters. The mode lines define combinations of horizontal and vertical timing that can produce a given resolution and refresh rate. For instance, a particular mode line might define a 1024×768 display at a 90Hz refresh rate, and another might represent 1024×768 at 72Hz. The upcoming section "Fine-Tuning Video Modes" describes how to modify your monitor's mode line without recomputing the whole thing from scratch. The preceding example splits the mode line across multiple lines, but more frequently it appears as a single line in the file.

Some mode lines represent video modes that are outside the horizontal or vertical sync ranges of a monitor. X can compute these cases and discard the video modes that a monitor can't support. If asked to produce a given resolution, X searches all the mode lines that accomplish the job, discards those that the monitor can't handle, and uses the remaining mode line that creates the highest refresh rate at that resolution. (If no mode line supports the requested resolution, X drops down to another specified resolution, as described shortly, and tries again.)

As a result of this arrangement, you'll see a large number of `Modeline` entries in the `XF86Config` file for XFree86 3.3.6 and earlier. Most end up going unused because they're for resolutions you don't use or because your monitor can't support them. You can delete these unused mode lines, but it's usually not worth the effort.

XFree86 4.*x* and X.org-X11 support a feature known as *Data Display Channel (DDC).* This is a protocol that enables monitors to communicate their maximum horizontal and vertical refresh rates and appropriate mode lines to the computer. The **XFree86 -configure** or **Xorg -configure** command uses this information to generate mode lines, and on every start,

the system can obtain horizontal and vertical refresh rates. The end result is that an XFree86 4.*x* or X.org-X11 system can have a substantially shorter `Monitor` section than is typical with XFree86 3.3.*x*.

Setting the Video Card

Your monitor is usually the most important factor in determining your maximum refresh rate at any given resolution, but X sends data to the monitor only indirectly, through the video card. Because of this, it's important that you be able to configure this component correctly. An incorrect configuration of the video card is likely to result in an inability to start X.

Choosing the Server or Driver

XFree86 4.*x* and X.org-X11 use driver modules that are stored in separate files from the main X server executable. The server can't determine what module is required automatically, however. Instead, you must give it that information in the `XF86Config` or `xorg.conf` file. In particular, the driver module is set by a line in the `Device` section, which resembles the following:

```
Driver "r128"
```

This line sets the name of the driver. The drivers reside in the `/usr/X11R6/lib/modules/drivers/` directory. Most of the drivers' filenames end in `_drv.o`, and if you remove this portion, you're left with the driver name. For instance, `r128_drv.o` corresponds to the `r128` driver.

> **NOTE**   The `xf86cfg` utility provides a large list of chipsets and specific video card models, so you can select the chipset or board from this list to have the utility configure this detail.

Setting Card-Specific Options

The `Device` section of the `xorg.conf` file sets various options related to specific X servers. A typical `Device` section resembles the following:

```
Section "Device"
    Identifier  "Videocard0"
    Driver      "r128"
    VendorName  "ATI"
    BoardName   "Rage 128"
    VideoRam    32768
EndSection
```

The `Identifier` line provides a name that's used in the subsequent `Screen` section to identify this particular `Device` section. (X configuration files frequently host multiple `Device` sections—for instance, one for a bare-bones VGA driver and one for an accelerated driver.) The `VendorName` and `BoardName` lines provide information that's useful mainly to people reading the file.

The VideoRam line is unnecessary with many servers and drivers because the driver can detect the amount of RAM installed in the card. With some devices, however, you may need to specify the amount of RAM installed in the card, in kilobytes. For instance, the preceding example indicates a card with 32MB of RAM installed.

Many drivers support additional driver-specific options. They may enable support for features such as hardware cursors (special hardware that enables the card to handle mouse pointers more easily) or caches (using spare memory to speed up various operations). Consult the XF86Config or xorg.conf man page or other driver-specific documentation for details.

Setting the Resolution and Color Depth

The Screen section tells X about the combination of monitors and video cards you're using. XFree86 4.*x* and X.org-X11 support multiple video cards and monitors on one system, and even in XFree86 3.3.6 and earlier you can define several monitors or video cards and switch between them by editing the Screen section. This could be handy if you're testing a new monitor or video card driver. In any event, the Screen section looks something like this:

```
Section "Screen"
    Identifier "Screen0"
    Device      "Videocard0"
    Monitor     "Monitor0"
    DefaultDepth 24
    SubSection "Display"
        Depth     24
        Modes     "1024x768" "1024x600" "800x600" "640x480"
    EndSubSection
    SubSection "Display"
        Depth     8
        Modes     "1024x768" "800x600" "640x480"
    EndSubSection
EndSection
```

The Device and Monitor lines refer to the Identifier lines in your Device and Monitor sections, respectively. The Screen section includes one or more Display subsections, which define the video modes that X may use. This example creates two such displays. The first uses a color depth of 24 bits (Depth 24) and possible video mode settings of 1024x768, 1024x600, 800x600, and 640x480. (These video modes are actually names that refer to the mode lines defined in the Monitor section, or to standard mode lines.) The second possible display uses an 8-bit color depth (Depth 8) and supports only 1024x768, 800x600, and 640x480 video modes.

To choose between the Display subsections, you include a DefaultDepth line. (This line is called DefaultColorDepth in XFree86 3.3.6 and earlier.) In this example, X uses the 24-bit display if possible, unless overridden by other options when starting X.

Graphical video modes require a certain amount of RAM on the video card itself. (On some laptop computers and computers with video hardware integrated into the motherboard,

a portion of system RAM is reserved for this use by the BIOS.) The total amount of RAM required is determined by an equation:

$$R = xres \times yres \times bpp \div 8{,}388{,}608$$

In this equation, R is the RAM in megabytes, $xres$ is the x resolution in pixels, $yres$ is the y resolution in pixels, and bpp is the bit depth. For instance, consider a 1280×1024 display at 32-bit color depth:

$$R = 1280 \times 1024 \times 24 \div 8{,}388{,}608 = 3.75\text{MB}$$

All modern video cards have at least 8MB of RAM, usually much more. This is more than enough to handle even very high resolutions at 32-bit color depth (the greatest depth possible). Thus, video RAM shouldn't be a limiting factor in terms of video mode selection, at least not with modern video hardware. Very old video cards can impose limits, so you should be aware of those limits.

> **NOTE** Modern video cards ship with large amounts of RAM to support 3D acceleration features. X supports such features indirectly through special 3D acceleration packages, but 3D acceleration support is limited compared to basic video card support. If 3D acceleration is important to you, you should research the availability of this support.

EXERCISE 5.1

Changing the X Resolution and Color Depth

In this lab, you will adjust the X resolution and color depth by editing the XF86Config or xorg.conf file. To do so, follow these steps:

1. Log onto the Linux system as a normal user.

2. Launch an xterm from the desktop environment's menu system.

3. Acquire root privileges. You can do this by typing **su** in an xterm, by selecting Session ➢ New Root Console from a Konsole, or by using sudo (if it's configured) to run the following commands.

4. Back up the current xorg.conf file by typing **cp /etc/X11/xorg.conf /root**. (Some distributions may place this file in /etc rather than /etc/X11. If you're using XFree86 rather than X.org-X11, the file is called XF86Config or XF86Config-4. You may need to adjust this command as appropriate.)

5. Load the xorg.conf file into a text editor by typing the editor's name followed by the complete path to the file in the xterm. For instance, **gedit /etc/X11/xorg.conf** loads the file into the gedit editor, which is part of GNOME. Other popular editor names include emacs, vi, kedit, and nedit.

EXERCISE 5.1 *(continued)*

6. Locate the Screen section of the file, identified by a line that reads Section "Screen".

7. Within the Screen section, locate a line that begins DefaultDepth. This line sets the default color depth of the display, in bits. Adjust the value to one you want to use. Common values include 8 (256 colors), 15 (32,768 colors), 16 (65,536 colors), 24 (16,777,216 colors), and 32 (over 4 million colors). Note that greater color depths often look better but may slow down the display.

8. Examine the Display subsections, which begin with the line Subsection "Display". If a Display subsection doesn't exist for the color depth you've selected, create one by copying an existing Display subsection for another color depth and changing the Depth line to match the color depth you selected. Copy all the lines from the Subsection "Display" line to the matching EndSubSection line.

9. In the Display subsection for the color depth you've selected, locate the Modes line. This line specifies the display modes (resolutions) that X will try to use, in order. A typical Modes line lists one or more display modes, in order, as in Modes "1280x1024" "1024x768" "800x600". In this case, X will try a 1280 × 1024 display first. If it can't produce that resolution, X will try a 1024 × 768 display. If X can't produce that display, it tries 800 × 600.

10. Adjust the Modes line so that your desired resolution is first in the list. If you want to have fallback resolutions, list them after the primary resolution. Be sure to enclose your mode in quote marks. (You can't list arbitrary display modes; for instance, "700x500" is unlikely to work. X should recognize common display resolutions, though.)

11. Save the changes to the file and exit from your editor.

12. If X isn't started, try starting it by typing **startx** at a text-mode command prompt.

13. If X is already running, log out of your session and, if necessary, select an option to restart X or press Ctrl+Alt+Backspace to force X to quit and restart.

Putting It All Together

XFree86 4.*x* and X.org-X11 require a section that's not present in the XFree86 3.3.6 configuration file: ServerLayout. This section links together all the other components of the X configuration:

```
Section "ServerLayout"
    Identifier      "single head configuration"
    Screen          "Screen0" 0 0
    InputDevice     "Mouse0" "CorePointer"
    InputDevice     "Keyboard0" "CoreKeyboard"
EndSection
```

Typically, this section identifies one `Screen` section and two `InputDevice` sections (for the keyboard and the mouse). Other configurations are possible, though. For instance, XFree86 4.*x* and X.org-X11 support *multi-head displays,* in which multiple monitors are combined to form a larger desktop than either one alone would support. In these configurations, the `ServerLayout` section would include multiple `Screen` sections.

Fine-Tuning Video Modes

Earlier, in "Setting the Monitor," I described X mode lines. These lines tell X about the video timings required to render particular video modes. In addition to the actual resolution and refresh rate, mode lines affect the positioning of the image on the monitor, the image's width, and the image's height. Sometimes, you might want to adjust these details. With modern monitors, the simplest approach is usually to use the monitor's controls; however, you might prefer to do it in software. A software solution will enable you to match the video modes between Linux and another OS more precisely or adjust the video display details for old monitors with insufficient front-panel controls. The tool to do all of this is called `xvidtune`, and it ships as part of X.

> You're most likely to want to run `xvidtune` when you use a *cathode ray tube (CRT)* monitor, which uses a bulky glass tube for the display. The slimmer *liquid crystal display (LCD)* technology maps pixels in a one-to-one manner, and LCD displays either use digital interfaces that don't require tuning with `xvidtune` or lock onto analog signals in such a way that `xvidtune` is unlikely to have any beneficial effect. If an LCD monitor can't seem to lock onto an analog signal properly, though, using `xvidtune` might help.

To use `xvidtune`, type its name in an `xterm` or other command prompt window. (Do not run `xvidtune` through a desktop environment or window manager's program-launching tool; the program displays some data to standard output—the console from which it's launched—and this will be inaccessible if you don't run the program from a command prompt.) The running `xvidtune` should resemble Figure 5.3.

F I G U R E   5 . 3   The `xvidtune` utility helps you tweak your mode lines for your monitor.
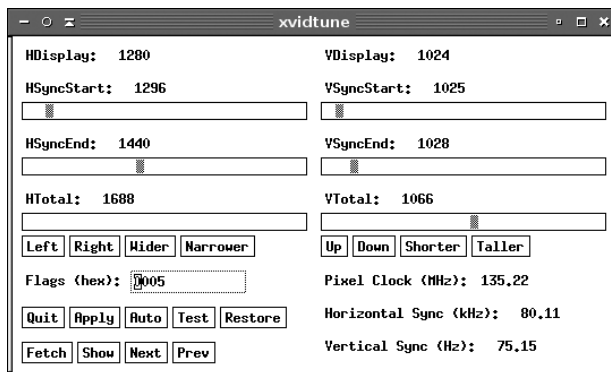
Using xvidtune entails clicking its buttons: Left, Right, Wider, and Narrower to control the horizontal position and size; and Up, Down, Shorter, and Taller to control the vertical position and size. These changes won't be immediately apparent, but clicking Test should temporarily implement them so that you can see the effect.

> Some X configurations disallow the sort of temporary changes that xvidtune uses. Specifically, if you see a line in XF86Config or xorg.conf that reads Option "DisableVidModeExtension" "yes", you may need to change yes to no and restart X in order to see the xvidtune changes.

Once you've made changes that are satisfactory, click Show. The result should be a new mode line on the xterm from which you launched the program. Copy this mode line to your XF86Config or xorg.conf file's Monitor section. You must add the Modeline keyword to the start of the line, and you may want to change the video mode's name to something distinctive—say, changing 1280x1024 to my1280x1024. You can then change your Screen section's Modes line to refer to the new mode and restart X.

Configuring X Fonts

Fonts have long been a trouble spot for Linux (or more precisely, for X). X was created at a time when available font technologies were primitive by today's standards, and although X has been updated in various ways to take advantage of newer technologies, these updates have been lacking compared to the font subsystems in most competing OSs. X's core font system can be configured from the X configuration file. Alternatively, you can configure a *font server*—a program that delivers fonts to one or many computers using network protocols—to handle the fonts. The latest Linux font technology sets up fonts in a way that's more independent of X and that produces more pleasing results, at least to most peoples' eyes.

Font Technologies and Formats

Font technologies can be classified as falling into one of two broad categories:

Bitmap fonts   The simplest type of font format is the *bitmap font,* which represents fonts much like bitmap graphics, in which individual pixels in an array are either active or inactive. Bitmap fonts are fairly easy to manipulate and display, from a programming perspective, which makes them good for low-powered computers. The problem is that each font must be optimized for display at a particular resolution. For instance, a font that's 20 pixels high will appear one size on the screen (typically 72 to 100 dots per inch, or dpi) but will be much smaller when printed (typically at 300 to 600 dpi). Similarly, you need multiple files to display a single font at multiple sizes (such as 9 point versus 12 point.) This means that a single font, such as Times, requires potentially dozens of individual files for display at different sizes and on different display devices. If you lack the correct font file, the result will be an ugly scaled display.

Outline fonts   Most modern fonts are distributed as *outline fonts* (aka *scalable fonts*). This type of format represents each character as a series of lines and curves in a high-resolution matrix. The computer can scale this representation to any font size or for any display resolution, enabling a single font file to handle every possible use of the font. The main problem with outline fonts is that this scaling operation is imperfect; scalable fonts often look slightly worse than bitmap fonts. Scaling and displaying the fonts also takes more CPU time than displaying a bitmap font. This factor used to be important, but on modern CPUs it's less of an issue.

Both bitmap and outline fonts come in several different formats. X ships with a number of basic bitmap and outline fonts, and you're unlikely to need to deal explicitly with bitmap fonts or their formats, so I don't describe them in any detail. Outline fonts are another matter, though. The two main formats are Adobe's *PostScript Type 1* (or Type 1 for short) and Apple's *TrueType.* Fonts available on the Internet and on commercial font CDs come in one or both of these formats.

XFree86 3.3.6 and earlier supported Type 1 fonts but not TrueType fonts. XFree86 4.*x* and X.org-X11 support both Type 1 and TrueType fonts. If you want to use TrueType fonts with XFree86 3.3.6 or earlier, you must do so with the help of a font server, as described shortly, in "Configuring a Font Server."

Configuring X Core Fonts

X core fonts are those that are handled directly by X. To configure these fonts, you must do two things: prepare a font directory that holds the fonts and add the font directory to X's font path.

Preparing a Font Directory

The first step to installing fonts is to prepare a directory in which to store them. XFree86 has traditionally stored its fonts in subdirectories of `/usr/X11R6/lib/X11/fonts/`, but X.org-X11 changes this to `/usr/share/fonts`. In either case, if you're adding fonts you've downloaded from the Internet or obtained from a commercial font CD-ROM, you might want to store your fonts elsewhere, such as `/opt/fonts` or `/usr/local/fonts`. (Chapter 4, "Managing Files and Filesystems," includes information on the logic behind Linux's directory system.) You may want to create separate subdirectories for fonts in different formats or from different sources.

When you're installing Type 1 fonts, Linux needs the font files with names that end in `.pfa` or `.pfb`; these files contain the actual font data. (The `.pfa` and `.pfb` files store the data in slightly different formats, but the two file types are equivalent.) Additional files distributed with Type 1 fonts aren't necessary for Linux. TrueType fonts come as `.ttf` files, and that's all you need for Linux.

> **NOTE**   Linux uses fonts in the same format that Windows, OS/2, and most other OSs use. Mac OS uses font files in special Macintosh-only "suitcases," which Linux cannot use directly. If you want to use such fonts in Linux, you must convert them. The FontForge program (`http://fontforge.sourceforge.net`) can do this conversion, among other things.

Once you've copied fonts to a directory, you must prepare a summary file that describes these fonts. This file is called `fonts.dir`, and it begins with a line that specifies how many fonts are described. Subsequent lines provide a font filename and an *X logical font description (XLFD)*, which is a rather tedious-looking description of the font. A complete `fonts.dir` line can look rather intimidating:

```
courb.pfa -ibm-Courier-bold-r-normal--0-0-0-0-m-0-iso8859-1
```

Fortunately, you needn't create this file manually; programs exist to do so automatically. In XFree86 4.3 and later and in X.org-X11, the simplest way to do the job is to use `mkfontscale` and `mkfontdir`:

```
# mkfontscale
# mkfontdir
```

The `mkfontscale` program reads all the fonts in the current directory and creates a `fonts.scale` file, which is just like a `fonts.dir` file but describes only outline fonts. The `mkfontdir` program combines the `fonts.scale` file with the `fonts.dir` file, creating it if it doesn't already exist.

Other programs to perform this task also exist. Most notably, `ttmkfdir` creates a `fonts.dir` file that describes TrueType fonts, while `type1inst` does the job for Type 1 fonts. The `mkfontscale` program is preferable because it handles both font types, but if you're using an older distribution that lacks this program, or if it's not doing a satisfactory job, you might try one of these alternative programs.

Adding Fonts to X's Font Path

Once you've set up fonts in a directory and created a `fonts.dir` file describing them, you must add the fonts to the X font path. You do this by editing the `Files` section of the `XF86Config` or `xorg.conf` file:

```
Section "Files"
    FontPath  "/usr/share/fonts/100dpi:unscaled"
    FontPath  "/usr/share/fonts/Type1"
    FontPath  "/usr/share/fonts/truetype"
    FontPath  "/usr/share/fonts/URW"
    FontPath  "/usr/share/fonts/Speedo"
    FontPath  "/usr/share/fonts/100dpi"
EndSection
```

> **NOTE** If your `Files` section contains `FontPath` lines that refer to `unix:/7100` or `unix:/-1` but that don't list conventional directories, read the section "Configuring a Font Server." You may want to modify your font server configuration rather than change the X core fonts directly.

To add your new font directory to the font path, duplicate one of the existing `FontPath` lines and change the directory specification to point to your new directory. The order of these directories is significant; when matching font names, X tries each directory in turn, so if two directories hold fonts of the same name, the first one takes precedence. Thus, if you want your new fonts to override any existing fonts, place it at the top of the list; if you want existing fonts to take precedence, add your directory to the end of the list.

> The `:unscaled` string in the first entry in the preceding example tells X to use bitmap fonts from this directory only if they exactly match the requested font size. Without this string, X will attempt to scale bitmap fonts from a font directory (with poor results). Typically, bitmap directories are listed twice, once near the top of the font path with the `:unscaled` specification and again near the bottom of the list without it. This produces quick display of matching bitmapped fonts, followed by any matching scalable fonts, followed by scaled bitmap fonts.

Once you've added your font directory to X's font path, you should test the configuration. The safest way to do this is to shut down X and restart it. (If your system boots directly into X, consult "Running an XDMCP Server" for information on doing this.) A quicker approach, but one that presents some opportunity for error, is to add the font path to a running system by using the `xset` program:

```
$ xset fp+ /your/font/directory
$ xset fp rehash
```

The first of these commands adds */your/font/directory* to the end of the font path. (Substitute +fp for `fp+` to add the directory to the start of the existing font path.) The second command tells X to re-examine all the font directories to rebuild the list of available fonts. The result is that you should now be able to access the new fonts. (You'll need to restart any programs that should use the new fonts, though.) One program to quickly test the matter is `xfontsel`. This program enables you to select an X core font for display so you can check to be sure that the fonts you've added are available and display as you expect.

Configuring a Font Server

Prior to the release of XFree86 4.0, several Linux distributions began using TrueType-enabled font servers to provide TrueType font support. These distributions, which include Fedora, Mandriva (formerly Mandrake), and Red Hat, continue to use a font server for local font delivery.

Font servers are also handy ways to deliver fonts to many computers from a central location. This can be a great time-saver if you want to add fonts to many computers—set them up to use

a font server and then tweak that server's font configuration. To use a font server, X must list that server in its font path:

```
Section "Files"
    FontPath  "unix:/7100"
    FontPath  "tcp/fount.pangaea.edu:7100"
EndSection
```

The first line in this example specifies a local font server. (Mandriva's default configuration uses unix:/-1 rather than unix:/7100.) The second line specifies that the font server on the remote system fount.pangaea.edu is to be used. If your system is already configured to use a font server, you needn't change the X configuration to add or delete fonts; instead, you can modify the font server's configuration. (You *can* still modify the X font configuration directly, but it may be cleaner to manage all the local fonts from one configuration file.)

To add fonts to a font server, you should first install the fonts on the system, as described earlier in "Preparing a Font Directory." You should then modify the font server's configuration file, /etc/X11/fs/config. Rather than a series of FontPath lines, as in the main X configuration file, the font server's configuration lists the font path using the catalogue keyword as a comma-delimited list:

```
catalogue = /usr/share/fonts/100dpi:unscaled,
            /usr/share/fonts/Type1,
            /usr/share/fonts/truetype,
            /usr/share/fonts/URW,
            /usr/share/fonts/Speedo,
            /usr/share/fonts/100dpi
```

The catalogue list may span several lines or just one. In either event, all of the entries are separated by commas, but the final entry ends without a comma. You can add your new font directory anywhere in this list.

Once you've saved your changes, you must restart the font server. Typically, this is done via SysV startup scripts (described in more detail in Chapter 6):

```
# /etc/init.d/xfs restart
```

At this point, you should restart X or type **xset fp rehash** to have X re-examine its font path, including the fonts delivered via the font server.

Although X core fonts and font servers were once very important, most modern X applications now emphasize an entirely different font system: Xft. You can add the same fonts as both X core fonts and as Xft fonts, but the Xft configuration requires doing things in an entirely new way.

Configuring Xft Fonts

X core fonts (including fonts delivered via a font server) have several important drawbacks:

- X core fonts aren't easy to integrate between the screen display and printed output. This makes them awkward from the point of view of word processing or other applications that produce printed output.

- X core fonts are server based. This means that applications may not be able to directly access the font files because the fonts could be stored on a different computer than the application. This can exacerbate the printing integration problem.

- X core fonts provide limited or no support for kerning and other advanced typographic features. Again, this is a problem for word processing programs and other programs that must generate printed output.

- X core fonts don't support *font smoothing* (aka *anti-aliasing*). This technology employs gray pixels (rather than black or white pixels) along curves to create an illusion of greater resolution than the display can produce.

These problems are deeply embedded in the X core font system, so developers have decided to bypass that system entirely. The result is the Xft font system, which is based in part on the Freetype library (`http://www.freetype.org`), an open-source library for rendering TrueType and Type 1 fonts. Xft is a client-based system, meaning that applications access font files on the computer on which they're running. Xft also supports font smoothing and other advanced font features. Overall, the result is greatly improved font support. The cost, though, is that Linux now has *two* font systems: X core fonts and Xft fonts.

Fortunately, you can share the same font directories through both systems. If you've prepared a font directory as described earlier, in "Preparing a Font Directory," you can add it to Xft. Load the `/etc/fonts/local.conf` file into a text editor. Look for any lines in this file that take the following form:

```
<dir>/font/directory</dir>
```

If such lines are present, duplicate one of them and change it to point to your new font directory. If not, create such a line just before the `</fontconfig>` line. Be sure not to embed your new font directory specification within a comment block, though. Comments begin with a line that reads `<!--` and end with a line that reads `-->`.

> If you create a font directory that holds several subdirectories, you can add just the main directory to `local.conf`. For instance, if you created `/opt/fonts/tt` and `/opt/fonts/type1`, adding `/opt/fonts` to `local.conf` will be sufficient to access all the fonts you installed on the system.

Once you've made these changes, type **fc-cache** as `root`. This command causes Xft to run through its font directories and create index files. These files are similar to the `fonts.dir` file in principle, but the details differ. If you fail to take this step, you'll still be able to access these fonts, but each user's private Xft cache file will contain the lists of fonts. Generating these files can take some time, degrading performance.

To test your Xft fonts, use any Xft-enabled program. The `gnome-font-properties` program is a handy one for quickly checking that your fonts are installed.

Managing GUI Logins

Linux can boot into a purely text-based mode in which the console supports text-based logins and text-mode commands. This configuration is suitable for a system that runs as a server computer or for a desktop system for a user who dislikes GUIs. Most desktop users, though, expect their computers to boot into a friendly GUI. For such users, Linux supports a GUI login system that starts X automatically and provides a GUI login screen. Configuring and managing this system requires you to understand a bit of how the system works, how to run it, and how to change the configuration.

The X GUI Login System

As described later in this chapter, in "Using X for Remote Access," X is a network-enabled GUI. This fact has many important consequences, and one of these relates to Linux's GUI login system. This system employs a network login protocol, the *X Display Manager Control Protocol (XDMCP)*. To handle remote logins, an XDMCP server runs on a computer and listens for connections from remote computers' X servers. To handle local logins, an XDMCP server runs on a computer and starts the local computer's X server. The XDMCP server then manages the local X server's display—that is, it puts up a login prompt like that shown in Figure 5.4.

Three XDMCP servers are common on Linux: the X Display Manager (XDM), the KDE Display Manager (KDM), and the GNOME Display Manager (GDM). A few more exotic XDMCP servers are also available, but these three are the most important. As you might guess by their names, KDM and GDM are associated with the KDE and GNOME projects, respectively, but neither limits your choice of desktop environment. (The upcoming section "Configuring Linux to Run Your Desktop Environment" provides information on selecting which desktop environment you want to run.) Most Linux distributions run either GDM or KDM as the default XDMCP server, but you can change which one your system uses if you don't like the default.

F I G U R E   5 . 4   An XDMCP server manages local GUI logins to a Linux system.
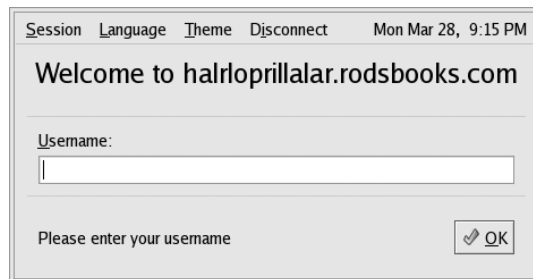
Running an XDMCP Server

Several methods exist to start an XDMCP server. The two most common are to launch it more or less directly from `init`, via an entry in `/etc/inittab`, or to launch it as part of a runlevel's startup script set, via a SysV startup script. Chapter 6 describes both `init` and SysV startup scripts in general, so consult it for information on these processes.

Whichever method is used, most distributions configure themselves to run their chosen XDMCP server when they start up in runlevel 5 but not when they start in runlevel 3. In fact, this is the only difference between these two runlevels in most cases. Thus, changing from runlevel 3 to runlevel 5 starts X and the XDMCP server on most distributions, and switching back to runlevel 3 stops X and the XDMCP server. As described in more detail in Chapter 6, you can change runlevels as `root` with the `telinit` command:

```
# telinit 5
```

Permanently changing the runlevel requires editing the `/etc/inittab` file and, in particular, its `id` line:

```
id:5:initdefault:
```

Change the number (5 in this case) to the runlevel you want to use as the default.

A few distributions—most notably Debian and Gentoo—attempt to start an XDMCP server in all runlevels (or don't do so at all). This is done through the use of a SysV startup script called `xdm`, `kdm`, or `gdm`. Thus, you can temporarily start or stop the XDMCP server by running this script and passing it the `start` or `stop` option. To permanently enable or disable the XDMCP server, you should adjust your SysV startup scripts, as described in Chapter 6.

In addition to the question of whether to run an XDMCP server is the question of *which* XDMCP server to run. Some distributions, such as Fedora, Mandriva, and Red Hat, use a script called `prefdm` to do the job. Instead of launching the XDMCP server directly, the system runs `prefdm`, which loads `/etc/sysconfig/desktop` to determine which XDMCP server to run. Specifically, the `DESKTOP` line sets the default desktop environment (KDE or GNOME, most commonly), which in turn sets the default XDMCP server. Debian and some other distributions use the choice of SysV startup script (`xdm`, `kdm`, or `gdm`) as a way to determine which XDMCP server is run.

Configuring an XDMCP Server

XDMCP servers, like most programs, can be configured. Unfortunately, this configuration varies from one server to another, although there are some commonalities.

Configuring XDM

XDM is the simplest of the major XDMCP servers. It accepts usernames and passwords but doesn't enable users to perform other actions, such as choose which desktop environment to run. (This must be configured through user login files, as described later in "Configuring Linux to Run Your Desktop Environment.")

XDM's main configuration file is /etc/X11/xdm/xdm-config. Most distributions ship with a basic xdm-config file that should work fine for a local workstation. If you want to enable the system to respond to remote login requests, or if you want to verify that the system is *not* so configured, you should pay attention to this line:

```
DisplayManager.requestPort: 0
```

This line tells XDM to not access a conventional server port. To activate XDM as a remote login server, you should change 0 to 177, the traditional XDMCP port. You must then restart XDM.

The /etc/X11/xdm/Xaccess file is another important XDM configuration file. If XDM is configured to permit remote access, this file controls who may access the XDM server, and in what ways. A wide-open system will contain lines that use an asterisk (*) to denote that anybody may access the system:

```
*
* CHOOSER BROADCAST
```

The first line tells XDM that anybody may connect, while the second one tells XDM that anybody may request a chooser—a display of local systems that accept XDMCP connections. To limit the choices, you should list individual computers or groups of computers instead of using the asterisk wildcard:

```
*.pangaea.edu
tux.example.com
*.pangaea.edu CHOOSER BROADCAST
```

This example lets any computer in the pangaea.edu domain connect or receive a chooser, and it also lets tux.example.com connect but not receive a chooser.

Many additional options are set in the /etc/X11/xdm/Xresources file, which hosts X resources, which are similar to environment variables but apply only to X-based programs. For instance, you can change the text displayed by XDM by altering the xlogin*greeting resource in this file.

Configuring KDM

KDM is based partly on XDM and so shares many of its configuration options. Unfortunately, the location of the KDM configuration files is unpredictable; sometimes KDM uses the XDM configuration files, other times they're stored in /etc/X11/kdm, and sometimes they're stored in a truly strange location such as /usr/kde/3.3/share/config/kdm/.

> If you can't find the KDM configuration files, try using your package management tools, described in Chapter 2, "Managing Software." Try obtaining lists of files in the kdebase package, or other likely candidates, and look for the KDM configuration files.

KDM expands on XDM by enabling users to select a session type when they log in, to shut down the computer from the main KDM prompt, and so on. Most of these extra options are set in the kdmrc file, which appears in the same directory as the other KDM configuration files. Some of these options override the more common XDM configuration options for the same features. In particular, the [Xdmcp] section provides options relating to network operation. The enable option in that section should be set to true if you want to support network logins.

Configuring GDM

GDM is more of a break from XDM than is KDM. GDM doesn't use the conventional XDM configuration files or similar files. Instead, it uses configuration files that are usually stored in /etc/X11/gdm. The most important of these files is gdm.conf, and it has a format similar to the kdmrc file. As with KDM, you should set the enable option to yes in the [xdmcp] section if you want to enable remote logins.

> A GUI control tool for GDM exists. Type **gdmconfig** as root to launch this program, which enables you to set GDM options using a point-and-click interface.

Like KDM, GDM provides extra options over those of XDM. These options include the ability to choose your login environment and shut down the computer. GDM is a bit unusual in that it prompts for the username and only then presents a prompt for the password. (Figure 5.4 shows the GDM username prompt.) XDM and KDM both present fields for the username and password simultaneously.

Configuring a Desktop Environment

Getting X running and displaying a login prompt are important for basic X use, of course, but beyond these steps lie various configuration options. Linux supports many different *desktop environments*—sets of programs that provide tools that integrate with one another for a unified experience. The desktop environment you pick will be important in determining how your system performs. (Desktop environment choice is a matter for individual users; each user can select a different desktop environment in a multi-user system, although of course the system administrator can limit choices by installing only some desktop environments.) Beyond picking a desktop environment, you must be able to get it to run and to configure certain important X environment options.

Desktop Environment Choices

Prior to the late 1990s, true desktop environments were rare in Linux. Users typically used bare *window managers*—programs that provide decorative and functional borders around windows and that manage the screen as a whole. Users would then add programs that could be easily launched from these window managers, such as file managers, calendar programs, mail readers, and so on.

Over time, fuller suites of programs have emerged. These desktop environments include window managers and all the extra programs in one package. Desktop environments also add integration between programs. For instance, programs might share a common address book, as well as "look-and-feel" settings such as default font options. Common desktop environments include:

GNOME   The *GNU Network Object Model Environment (GNOME)* is the favored desktop environment in Fedora, Red Hat, and some other distributions. It's a very large suite of programs that's built atop the GIMP Tool Kit (GTK+) development library, and it has a tendency to assimilate originally unaffiliated programs, adding their technological distinctiveness to its own. You can learn more at `http://www.gnome.org`.

KDE   The *K Desktop Environment (KDE)* is built around the Qt widget set. It tends to rely less on assimilating other projects than does GNOME, but the two are similar in overall capabilities. KDE is the favored desktop environment in Mandriva, Slackware, SuSE, and some other distributions. You can learn more at `http://www.kde.org`.

XFce   GNOME and KDE are the dominant desktop environments in Linux, but both are huge and resource hungry. XFce (`http://www.xfce.org`) is a much slimmer environment. Although it's based on the same GTK+ widget set as GNOME, XFce is very different. It's a good choice for those with weaker systems because it puts less of a load on the system than either GNOME or KDE.

In addition to these major choices, several less popular desktop environments exist, such as XPde (`http://www.xpde.com`, a desktop environment designed to mirror the Windows XP "look and feel" as much as possible) and the Common Desktop Environment (CDE, a commercial desktop environment after which XFce is loosely modeled). Furthermore, you can always run a "bare" window manager in Linux, just as you could in the past. You can then build up options by using the window manager's configuration files or your X login files, as described in the next section. Consult `http://xwinman.org` for pointers to many popular (and even more less-popular) window managers.

Most Linux distributions ship with GNOME, KDE, and XFce, but some (particularly smaller distributions) omit at least one of these. If one isn't installed but you want to install it, you can install it from your distribution's installation medium or from files downloaded from the environment's website.

Configuring Linux to Run Your Desktop Environment

Installing a desktop environment or window manager is just a prerequisite for getting it to run. Both GDM and KDM provide tools to enable users to select their desktop environment when they log in, but you must ensure that the environment is available in the GDM or KDM menu. For systems that run XDM or that start up in text mode, editing user configuration files will determine which desktop environment is run. (Some GDM and KDM options also rely on user configuration files.)

Configuring GDM and KDM Sessions

Modern versions of both GDM and KDM rely on files stored in `/usr/share/xsessions` to determine what options to present for possible login sessions. This directory holds files with names of the form *environment*`.desktop`, where `environment` is the environment name (such as `KDE` or `GNOME`). The precise names vary from one distribution to another, but you should be able to spot the session file for your environment of choice—if it exists.

If a session file doesn't exist for your environment, you can create one. The simplest way to do this is probably to copy an existing file and make appropriate changes. Listing 5.1 shows a sample file. You should change the `Name` and `Comment` entries, which present a name and a comment, both of which are visible to users. Just as important, you should change the `Exec` and `TryExec` lines, which point to the program that's used to launch the environment. KDE uses `startkde`, GNOME uses `gnome-session`, XFce uses `xfce-session`, and most bare window managers use their own names. (Listing 5.1's IceWM window manager is an exception; it uses `icewm-session`.)

Listing 5.1: Sample X Session File for GDM and KDM

```
[Desktop Entry]
Encoding=UTF-8
Name=IceWM
Comment=This session logs you into IceWM
Exec=icewm-session
TryExec=icewm-session
# no icon yet, only the top three are currently used
Icon=
Type=Application
```

> **NOTE** Most X session files include a large number of `Name` and `Comment` lines, each of which includes a language code, such as `Name[fr]` for French. You can delete all of these except for the language your system actually uses or even just leave the bare `Name` and `Comment` lines, as in Listing 5.1.

Once you've added an X session file, you must restart your XDMCP server, as described earlier in "Running an XDMCP Server." When KDM or GDM starts up again, you should see your new session in its list of available sessions and be able to select it. Ordinary users will be able to do the same.

Many distributions remember users' choices for login sessions and use those choices as their defaults or give users the option of changing their defaults when they select a new session. Other distributions, though, including the popular Fedora and Red Hat, require the user to run a special program to make these changes explicit. For Fedora and Red Hat, this program is the Desktop Switching Tool (`switchdesk`).

Configuring Environments to Run from XDM and *startx*

If your system runs XDM or boots into text mode, scripts control the environment that's started, as summarized in Table 5.1. (GDM and KDM may also use the XDM scripts, depending on their configuration.) These scripts run any commands that should be run globally or by users for these login methods. Typically, the user startup script includes a call to the desktop environment or window manager that the user wants to run. It may also launch an xterm or other programs the user wants running at all times.

TABLE 5.1   Common X Startup Scripts

X Startup Method	Global Startup Script	User Startup Script
XDM	/etc/X11/xdm/Xsession	~/.xsession
startx	/etc/X11/xinit/xinitrc	~/.xinitrc

Chapter 6 describes scripting in more detail, but you can create an X login script without knowing much about scripting. Such scripts typically just launch a few programs, as shown in Listing 5.2. This listing begins with the string #!/bin/bash, which identifies the file as a script. Subsequent lines are commands, much like commands typed at a command prompt. The aplay commands play sounds using the Advanced Linux Sound Architecture (ALSA) sound system; xterm launches an xterm window; and xfce-session begins an XFce session.

Listing 5.2: A Typical X Login Script

```
#!/bin/bash
aplay /usr/local/sounds/login.wav &
xterm &
xfce-session
aplay /usr/local/sounds/logout.wav
```

One important detail of login scripts is that all the commands up to, but not including, the main desktop environment or window manager command (xfce-session in this example) should end with an ampersand (&). As described in Chapter 1, "Linux Command-Line Tools," the ampersand signals that the program is to be run in the background. In scripts, it means that execution should continue, without waiting for the program to terminate. Commands that don't end in an ampersand cause the script to stop running until the command completes. Thus, Listing 5.2 runs until the xfce-session command runs, then it pauses until XFce terminates—normally because the user has selected an exit option. The script then continues, playing the logout sound and ultimately exiting itself. If you omit the ampersand on the final aplay command, execution will pause while the sound plays, which will slow down the login process slightly. Omitting the ampersand on the xterm line, though, means that XFce won't launch until the user exits from the xterm window, which probably wasn't the desired result.

Once you've created the script, you must mark it as executable using chmod (described in more detail in Chapter 4):

```
$ chmod ~/.xinitrc
```

You should now be able to log into your environment using XDM or by typing **startx** after a text-mode login. (Of course, you must edit and adjust the correct login file for your login method.)

Configuring the X Environment

Traditionally, X programs have relied on X resources to control various user interface details, such as their default fonts, window colors, and so on. X resources are similar in concept to environment variables (described in Chapter 6), but they're used only by X-based programs and details of how they're controlled differ. In particular, X resources are set in users' ~/.Xresources and ~/.Xdefaults files. These files have the same format, and both are used to set X resources; the main difference is that .Xresources is consulted when the window manager starts up, whereas .Xdefaults can be used at any time. On some distributions, the /etc/X11/app-defaults directory holds files with resources that are applied globally. Whatever the name, resource files consist of lines that set resources by program name and resource name:

ProgName*ResName: *Value*

The *ProgName* is the program name (or more precisely, the name that the program uses to search for resources), the *ResName* is the resource name, and the *Value* is the value used for that resource. For instance, consider the following resources:

```
XTerm*Background:  linen
XTerm*Foreground:  black
XTerm*font:        9x15
XTerm*saveLines:   1000
XTerm*geometry:    +50+100
```

These resources set several xterm options, such as the background and foreground colors, the font, the number of lines to save in the xterm scrollback buffer, and the size of the window (in characters). Consult xterm's man page for further information on all of these resources.

Unfortunately, X resources aren't entirely standardized across applications; you must consult the documentation for each program to learn the names of the resources it supports and what the effects of various values are. Furthermore, many modern programs don't use X resources at all. Instead, they rely on their own configuration files and the options you set there. Increasingly, you can set these options using the programs' own user interfaces, which is an approach that's more familiar to those who have used Windows or Mac OS.

Configuring a Window Manager

Beyond the X environment, most desktop environments and window managers provide their own configuration files. (Some provide directories with multiple configuration files.) Like most program configuration files, these files are usually named after the programs they control, such as `~/.icewm` for IceWM, `~/.fvwm` for FVWM, or `.kde` for KDE.

Depending on the window manager or desktop environment, you may be able to make changes by directly editing the configuration file, by making changes using the environment's GUI tools and saving the changes from the GUI, or both. Details vary so much from one environment to another that you must consult environment-specific documentation for details. Indeed, even the features you can edit vary. Most window managers, though, provide the ability to change the programs you can launch from a program-launch menu at a bare minimum. You may also be able to configure virtual desktops, change the background color or image, add small programs to a menu bar, and more.

Using X for Remote Access

As noted earlier, in "The X GUI Login System," X is a network-enabled GUI. This fact enables you to run Linux programs remotely—you can set up a Linux system with X programs and run them from other Linux (or even non-Linux) computers. Similarly, you can use a Linux computer as an access terminal for X programs that run on a non-Linux Unix computer, such as one running Solaris. To do this, you should first understand something of X's network model—where the client and server systems are located, how X controls access to itself, and so on. You can then proceed to perform the remote accesses.

X Client/Server Principles

Most people think of servers as powerful computers hidden away in machine rooms and of clients as the desktop systems that ordinary people use. Although this characterization is often correct, it's very wrong when it comes to X. X is a server, meaning that the X server runs on the computer at which the user sits. X clients are the programs that users run—xterm, xfontsel, KMail, OpenOffice.org, and so on. In most cases, the X server and its clients reside on the same computer, so this peculiar terminology doesn't matter; however, when you use X for remote access, you must remember that the X server runs on the user's computer while the X clients run on the remote system.

To make sense of this peculiarity, think of it from the program's point of view. For instance, consider a web browser such as Mozilla Firefox. This program accesses web pages stored on a web server computer. The web server responds to requests to load files from Firefox. Just as Firefox loads files, it displays files on the screen and accepts input from its user. From the program's point of view, this activity is much like retrieving web pages, but it's handled by an X server rather than a web server. This relationship is illustrated in Figure 5.5.

FIGURE 5.5   From a program's point of view, the X server works much like a conventional network server such as a web server.
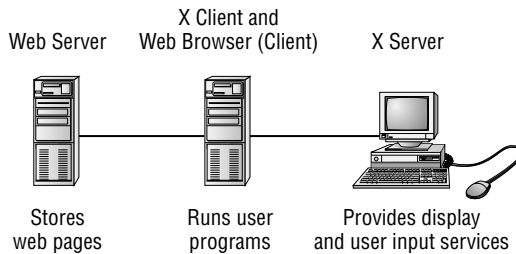
Ordinarily, Linux is configured in such a way that its X server responds only to local access requests as a security measure. Thus, if you want to run programs remotely, you must make some changes to have Linux lower its defenses—but not too far lest you let anybody access the X server, which could result in security breaches.

Using Remote X Clients

Suppose that your local network contains two machines. The computer called zeus is a powerful machine that hosts important programs, like a word processor and data analysis utilities. The computer called apollo is a much less powerful system, but it has an adequate monitor and keyboard. Therefore, you want to sit at apollo and run programs that are located on zeus. Both systems run Linux. To accomplish this task, follow these steps:

1. Log into apollo and, if it's not already running X, start it.
2. Open a terminal (such as an xterm) on apollo.
3. Type **xhost +zeus** in apollo's terminal. This command tells apollo to accept for display in its X server data that originates on zeus.
4. Log into zeus from apollo. You might use Telnet or Secure Shell (SSH), for instance. The result should be the ability to type commands in a shell on zeus.
5. On zeus, type **export DISPLAY=apollo:0.0**. (This assumes you're using bash; if you're using tcsh, the command would be **setenv DISPLAY apollo:0.0**.) This command tells zeus to use apollo for the display of X programs. (Chapter 6 describes environment variables, such as DISPLAY, in greater detail.)
6. Type whatever you need to type to run programs at the zeus command prompt. For instance, you could type **ooffice** to launch OpenOffice.org. You should see the programs open on apollo's display, but they're running on zeus—their computations use zeus's CPU, they can read files accessible on zeus, and so on.
7. After you're done, close the programs you've launched, log off zeus, and type **xhost -zeus** on apollo. This will tighten security so that a miscreant on zeus won't be able to modify your display on apollo.

Sometimes, you can skip some of these steps. For instance, depending on how it's configured, SSH can forward X connections, meaning that SSH intercepts attempts to display X information and passes those requests on to the system that initiated the connection. When this happens, you can skip steps 3 and 5, as well as the xhost command in step 7. (See the Real-World Scenario sidebar "Encrypting X Connections with SSH.")

As an added security system, many Linux distributions today configure X to ignore true network connections. If your distribution is so configured, the preceding steps won't work; when you try to launch an X program from the remote system, you'll get an error message. To work around this problem, you must make an additional change, depending on how X is launched:

GDM Check the GDM configuration file (typically /etc/X11/gdm/gdm.conf) and look for the line DisallowTCP=true and change it to read DisallowTCP=false.

KDM or XDM These two XDMCP servers both rely on settings in the Xservers file (in /etc/X11/xdm for XDM, and in this location or some other highly variable location for KDM). Look for the line that begins with :0. This line contains the command that KDM or XDM uses to launch the X server. If this line contains the string -nolisten tcp, remove that string from the line. This will eliminate the option that causes X to ignore conventional network connections.

X launched from a text-mode login If you log in using text mode and type **startx** to launch X, you may need to modify the startx script itself, which is usually stored in /usr/bin. Search this script for the string -nolisten tcp. Chances are this string will appear in a variable assignment (such as to defaultserverargs) or possibly in a direct call to the X server program. Remove the -nolisten tcp option from this variable assignment or program call.

Once you've made these changes, you'll need to restart X as described earlier in "Running an XDMCP Server." Thereafter, X should respond to remote access requests.

> **WARNING** Distribution maintainers disable X's ability to respond to remote requests for a reason. If X responds to remote network requests, the risk of an intruder using a bug or misconfiguration to trick users by displaying bogus messages on the screen is greatly increased. Thus, you should disable this protection only if you're sure it's necessary. You may be able to use a Secure Shell (SSH) link without disabling this protection. SSH has other benefits as well, as described shortly.

Another option for running X programs remotely is to use the Virtual Network Computing (VNC) system (http://www.realvnc.com). VNC runs a special X server on the computer that's to be used from a distance, and a special VNC client runs on the computer at which you sit. You use the client to directly contact the server. This reversal of client and server roles over the normal state of affairs with conventional X remote access is beneficial in some situations, such as when you are trying to access a distant system from behind certain types of firewall. VNC is also a cross-platform protocol; it's possible to control a Windows or Mac OS system from Linux using VNC, but this is not possible with X. (X servers for Windows and Mac OS are available, allowing you to control a Linux system from these non-Linux OSs.)

> ## 🌐 Real World Scenario
>
> ### Encrypting X Connections with SSH
>
> The SSH protocol is a useful remote-access tool. Although it's often considered a text-mode protocol, SSH also has the ability to *tunnel* network connections—that is, to carry another protocol through its own encrypted connection. This feature is most useful for handling remote X access. You can perform the steps described in "Using Remote X Clients" but omit steps 3, 5, and the `xhost` command in step 7. This greatly simplifies the login process and adds the benefits of SSH's encryption, which X doesn't provide. On the other hand, SSH's encryption is likely to slow down X access, although if you enable SSH's compression features, this problem might be reduced in severity. Overall, tunneling X through SSH is the preferred method of remote X access, particularly when any network in the process isn't totally secure.
>
> SSH tunneling does require certain options be set, though. In particular, you must either use the `-X` option to the `ssh` client program or set the `ForwardX11` option to yes in `/etc/ssh_config` on the client system. You must also set the `X11Forwarding` option to yes in the `/etc/sshd_config` file on the SSH server system. These options enable SSH's X forwarding feature; without these options, SSH's X forwarding won't work.

Summary

X is Linux's GUI system. In part because of Linux's modular nature, though, X isn't a single program; you have your choice of X servers to run on Linux. Fortunately, most Linux distributions use the same X server as all others (X.org-X11), although another one (XFree86) was the standard choice until 2004. Both of these servers are configured in much the same way, using the `xorg.conf` (for X.org-X11) or `XF86Config` configuration file. Whatever its name, this file consists of several sections, each of which controls one X subsystem, such as the mouse, the keyboard, or the video card. This file also controls X's core fonts system, but you can use a font server in addition to this system, and most modern programs are now emphasizing an entirely new font system, Xft, instead of X core fonts. For this reason, Linux font configuration can be complex.

X's GUI login system uses an XDMCP server, which starts X and manages the X display. Three XDMCP servers are in common use in Linux: XDM, KDM, and GDM. These all perform the same basic tasks, but configuration details differ. (XDM is also less sophisticated than KDM or GDM.) After the login process, the system launches a user's preferred window manager or desktop environment. The details of how this is done depend on the XDMCP server in use, but you may need to edit system or local configuration files to have the right programs launch.

X is a network-enabled GUI, which means you can use an X server to access programs running on another computer. Doing so requires performing a few steps for each login session. You can also tunnel X accesses through SSH, which greatly improves the security of the connection.

Exam Essentials

Name the major X servers for Linux. XFree86 has been the traditional standard Linux X server, but in 2004 X.org-X11 (which was based on XFree86) rapidly gained prominence as the new standard Linux X server. Accelerated-X is a commercial X server that sometimes supports video cards that aren't supported by XFree86 or X.org-X11.

Summarize the common X configuration tools. For XFree86 3.3.6, `xf86config`, `Xconfigurator`, and `XF86Setup` were the most popular configuration tools. For XFree86 4.*x* and X.org-X11, the X server itself, `Xconfigurator`, `xf86cfg/xorgcfg`, and distribution-specific tools are commonly used. In all cases, you can also edit the X configuration file manually using a text editor.

Describe the X configuration file format. The XFree86 and X.org-X11 configuration file is broken into multiple sections, each of which begins with the keyword `Section` and ends with `EndSection`. Each section sets options related to a single X feature, such as loading modules, specifying the mouse type, or describing the screen resolution and color depth.

Summarize the differences between X core fonts, a font server, and Xft fonts. X core fonts are managed directly by X, and they lack modern font features such as font smoothing. Font servers integrate with the X core fonts but run as separate programs and may optionally deliver fonts to multiple computers on a network. Xft fonts bypass the X core font system to provide client-side fonts in a way that supports modern features such as font smoothing.

Explain the role of an XDMCP server. An XDMCP server, such as XDM, KDM, or GDM, launches X and controls access to X via a login prompt—that is, it serves as Linux's GUI login system. XDMCP servers are also network enabled, providing a way to log in remotely from another X server.

Describe the methods users may employ to select a desktop environment or window manager. If the computer runs KDM or GDM, users may select a login environment from a menu in the login server—but the desired environment must be defined a file in the `/usr/share/xsessions` directory. For systems running XDM, the `~/.xsession` file is a login script that can be modified to launch the desired window manager or desktop environment. When X is started from text mode via `startx`, the `~/.xinitrc` file serves a function similar to that of `~/.xsession`.

Summarize X's client/server model. An X server runs on the user's computer to control the display and accept input from the keyboard and mouse. Client programs run on the same computer or on a remote computer to do the bulk of the computational work. These client programs treat the X server much as they treat other servers, requesting input from and sending output to them.

Explain the benefits of using SSH for remote X access. SSH can simplify remote X-based network access by reducing the number of steps required to run X programs from a remote computer. More important, SSH encrypts data, which keeps information sent between the X client and X server secure from prying eyes.

Review Questions

1. You want to reconfigure X.org-X11 after changing a computer's video card, so you type **XF86Setup** as `root` and enter the new information in the program's GUI. When you test the X server, though, you find that your changes had no effect. Why not?

   A. The XF86Setup program is designed to be run by an ordinary user, not by `root`.

   B. You selected the wrong video card in the XF86Setup program.

   C. You must copy the temporary configuration file created by XF86Setup to /etc/X11.

   D. XF86Setup is a configuration tool for XFree86 3.3.6 and earlier, not for X.org-X11.

2. When you configure an X server, you need to make changes to configuration files and then start or restart the X server. Which of the following can help streamline this process?

   A. Shut down X by switching to a runlevel in which X doesn't run automatically, and then reconfigure it and use `startx` to test X startup.

   B. Shut down X by booting into single-user mode, and then reconfigure X and use `telinit` to start X running again.

   C. Reconfigure X, and then unplug the computer to avoid the lengthy shutdown process before restarting the system and X along with it.

   D. Use the `startx` utility to check the X configuration file for errors before restarting the X server.

3. Which of the following summarizes the organization of the XF86Config and xorg.conf files?

   A. The file contains multiple sections, one for each screen. Each section includes subsections for individual components (keyboard, video card, and so on).

   B. Configuration options are entered in any order desired. Options relating to specific components (keyboard, video card, and so on) may be interspersed.

   C. The file begins with a summary of individual screens. Configuration options are preceded by a code word indicating the screen to which they apply.

   D. The file is broken into sections, one or more for each component (keyboard, video card, and so on). The file also has one or more sections that define how to combine the main sections.

4. A monitor's manual lists its range of acceptable synchronization values as 27–96kHz horizontal and 50–160Hz vertical. What implications does this have for the resolutions and refresh rates the monitor can handle?

   A. The monitor can run at up to 160Hz vertical refresh rate in all resolutions.

   B. The monitor can handle up to 160Hz vertical refresh rate depending on the color depth.

   C. The monitor can handle up to 160Hz vertical refresh rate depending on the resolution.

   D. The monitor can handle vertical resolutions of up to 600 lines (96,000 ÷ 160), but no more.

5. In what section of `XF86Config` or `xorg.conf` do you specify the resolution that you want to run?

   A. In the `Screen` section, subsection `Display`, using the `Modes` option

   B. In the `Monitor` section, using the `Modeline` option

   C. In the `Device` section, using the `Modeline` option

   D. In the `DefaultResolution` section, using the `Define` option

6. The following line appears in your X server's mouse configuration area. What can you conclude?

   `Option      "Protocol" "PS/2"`

   A. The mouse is connected to the PS/2 hardware mouse port.

   B. The mouse uses the PS/2 software communication standard.

   C. The computer is an ancient IBM PS/2 system.

   D. The mouse was designed for use with IBM's OS/2.

7. Which of the following sections is present in the X configuration files for XFree86 4.*x* and X.org-X11 but not for XFree86 3.3.6?

   A. `Monitor`

   B. `ServerLayout`

   C. `Pointer`

   D. `Screen`

8. What is an advantage of a font server?

   A. It provides faster font displays than is otherwise possible.

   B. It can simplify font maintenance on a network with many X servers.

   C. It's the only means of providing TrueType support for XFree86 4.*x*.

   D. It enables the computer to turn a bitmapped display into an ASCII text file.

9. What methods do Linux distributions use to start X automatically when the system boots? (Select all that apply.)

   A. Start an XDMCP server from the `Start` folder

   B. Start an XDMCP server from an `~/.xinitrc` script

   C. Start an XDMCP server via a SysV startup script

   D. Start an XDMCP server from `init`

10. You find a reference to the `prefdm` program in `/etc/inittab`. What does this program do?

   A. It reads `/etc/sysconfig/desktop` to determine which XDMCP server to run and then runs the XDMCP server.

   B. It's an XDMCP server that provides preferential login treatment to important users, such as `root`.

   C. It displays a dialog box that enables you to select which XDMCP server you want to run as the default.

   D. It runs before the `fdm` program, preparing data directories to hold important login variables.

11. How would you change the text displayed by XDM as a greeting?

    A. Click Configure ➢ Greeting from the XDM main menu and edit the text in the resulting dialog box.

    B. Edit the `/etc/X11/xdm/Xresources` file and change the text in the `xlogin*greeting` line.

    C. Edit the `/etc/X11/XF86Config` file and change the `Greeting` option in the `xdm` area.

    D. Run `xdmconfig` and change the greeting on the Login tab.

12. Which of the following features do KDM and GDM provide that XDM does not?

    A. An encrypted remote X-based access ability, improving network security

    B. The ability to accept logins from remote computers, once properly configured

    C. The ability to select the login environment from a menu on the main login screen

    D. A login screen that shows the username and password simultaneously rather than sequentially

13. You want to enable remote access in GDM. Which of the following is one step you would take in doing so?

    A. Edit the `/etc/X11/gdm/gdm-config` file and set the `DisplayManager.requestPort` option to `177`.

    B. Edit the `/etc/ssh/sshd_config` file's `X11Forwarding` line to read `yes`.

    C. Edit the `/etc/X11/gdm/gdm.conf` file and change the `enable` line in the `[xdmcp]` section to `yes`.

    D. You can't; this feature isn't supported by GDM, but it is supported by KDM and XDM.

14. Which of the following desktop environments is a commercial package?

    A. KDE

    B. GNOME

    C. CDE

    D. XFce

15. You want users to be able to select a new window manager you've installed from the GDM login screen. In what directory should you create a configuration file to add this option?

    A. `/var/spool/logins`

    B. `/etc/X11/gdm`

    C. `/usr/share/xsessions`

    D. `/etc/X11/xlogin`

16. The following `.xinitrc` script is present in a user's home directory, with the correct permissions. What will be the effect when the user types **startx** from a text-mode login?

```
#/bin/bash
xterm
startkde &
```

  A. The `xterm` program will run, and once it's started, KDE will start. The X session will terminate when the user exits from KDE, whereupon the `xterm` will also close.

  B. The `xterm` program will run and remain active, with no window manager, until the user terminates it. At that time, KDE will begin to start, but X will terminate before KDE can fully load.

  C. The `xterm` program will run, and once it's started, KDE will start. The X session will terminate when the user exits from *both* KDE and the `xterm` program; quitting either alone will not terminate X.

  D. The `xterm` program will run and immediately terminate, whereupon KDE will start. KDE will remain active until the user exits from it, whereupon X will also stop.

17. What file should you edit to change the desktop environment or window manager launched by XDM?

  A. `~/.bashrc`

  B. `~/.xsession`

  C. `~/.xinitrc`

  D. `~/.Xresources`

18. The X-based program MegaProg's menu font has changed to Times, but you prefer Helvetica in this role. The program's in-window font was, and should remain, Times. In investigating this problem, you discover several lines in your `~/.Xresources` file that seem suspicious. Which one should you edit to fix the problem?

  A. `MegaProg*mfont: Times`

  B. `MegaProg*body: Times`

  C. `MegaProg*font: Times`

  D. Possibly any or none of the above; consult the program's documentation.

19. Which of the following commands tells the X server to accept connections from `penguin.example.com`?

  A. **xhost +penguin.example.com**

  B. **export DISPLAY=penguin.example.com:0**

  C. **telnet penguin.example.com**

  D. **xaccess penguin.example.com**

20. What is an advantage of using SSH to tunnel an X session compared to using X's network features more directly?

   A. SSH supports a higher bit depth than do other network access methods.

   B. SSH supports font smoothing, which isn't possible with more direct connections.

   C. SSH provides compression features to fit larger displays on smaller monitors.

   D. SSH provides encryption features that improve the security of the connection.

Answers to Review Questions

1. D. You must use an X configuration program for the X server you use, and in this case, XF86Setup is the wrong tool for configuring X.org-X11, as stated in option D. X configuration programs can sometimes be run by ordinary users, but this isn't a requirement, and if they write their changes directly to the configuration file, they must be run as root, contrary to option A. Although it's possible that you selected the wrong video card, as stated in option B, this isn't the most important problem, and even if you selected the right card, this problem would occur. Some configuration tools do drop new configuration files in /root or some other directory, as suggested by option C, but again, such placement isn't the main problem in this scenario.

2. A. On most Linux systems, some runlevels don't run X by default, so using one of them along with the startx program (which starts X running) can be an effective way to quickly test changes to an X configuration. The telinit program changes runlevels, which is a lengthy process compared to using startx. Unplugging the computer to avoid the shutdown process is self-defeating since you'll have to suffer through a long startup (if you use a non-journaling filesystem), and it can also result in data loss. The startx utility doesn't check the veracity of an X configuration file; it starts X running from a text-mode login.

3. D. The XF86Config and xorg.conf file design enables you to define variants or multiple components and easily combine or recombine them as necessary.

4. C. The vertical refresh rate range includes a maximum value, but that value may be reduced when the resolution and vertical refresh rate would demand a higher horizontal refresh rate than the monitor can handle. In practice, it's usually the horizontal limit that's most important, at least when running at typical resolutions. The color depth is irrelevant to the computation.

5. A. Option A describes the correct location for this option. The Modeline option in the Monitor section (as described in option B) defines *one* possible resolution, but there are usually several Modeline entries defining many resolutions. The Modeline option doesn't exist in the Device section (as suggested by option C), however, nor is that section where the resolution is set. There is no DefaultResolution section (as referenced in option D).

6. B. "PS/2" can refer to both a hardware interface and a software protocol, but used in the context of the Protocol option, it unambiguously refers to the software protocol. Option A *might* be correct, but the specified line is insufficient evidence of that; USB mice generally use the PS/2 protocol or a variant of it, such as the Intellimouse PS/2 protocol. Although the PS/2 hardware port and protocol originated with the IBM PS/2 computer mentioned in option C, many other computers now use them. Mice that use the PS/2 protocol may be used with just about any OS, not just IBM's OS/2.

7. B. The ServerLayout section is unique to XFree86 4.*x* and its derivative, X.org-X11; this section was not present in XFree86 3.3.6 and earlier. The Monitor and Screen sections are present in all three X servers' configuration files, while the Pointer section is unique to XFree86 3.3.6.

8. B. By maintaining fonts on one font server and pointing other X servers to that font server, you can reduce the administrative cost of maintaining the fonts on all the systems. Font servers do not produce faster font displays than X's local font handling; if anything, the opposite is true. XFree86 4.*x* supports TrueType fonts directly, but XFree86 3.3.6 and earlier didn't include this support by default. Converting a bitmapped display into ASCII text is a function of optical character recognition (OCR) software, not a font server.

9. C, D. XDMCP servers are typically launched either from a SysV startup script or by `init` (as specified in `/etc/inittab`), as described in options C and D. The XDMCP server then starts X. The `Start` folder mentioned in option A is a Windows construct, not a Linux construct. The `~/.xinitrc` script mentioned in option B is an X login script used when starting X from the command line via `startx`; it's not used to automatically start X when the system boots.

10. A. Some distributions, such as Fedora and Red Hat, use `prefdm` as a front end to the XDMCP server, enabling `init` to call a single program (namely, `prefdm`), which in turn launches the specific XDMCP server the system administrator has configured by editing `/etc/sysconfig/desktop`. Options B, C, and D all describe completely fictitious programs.

11. B. The XDM greeting is a resource set in the `/etc/X11/xdm/Xresources` file, so option B is correct. XDM doesn't offer many options on its main screen and certainly not one to change its greeting, as described in option A. Although the `XF86Config` file mentioned in option C is real, this file provides no XDM configuration options because XDM is a separate program from XFree86. There is no standard `xdmconfig` program, as mentioned in option D.

12. C. KDM and GDM add many features, one of which is a menu that enables users to select their desktop environment or window manager when they log in rather than specifying it in a configuration file, as option C states. Option A describes one of the advantages of the Secure Shell (SSH) as a remote-access protocol. Option B describes a feature common to all three XDMCP servers. Option D describes the way both KDM and XDM function; GDM is the one that presents username and password fields in series rather than simultaneously.

13. C. Enabling remote access via GDM requires making the changes described in option C. Option A describes the equivalent change that's necessary in XDM (except for slight changes to the filename), but GDM doesn't use this method. Option B describes one change that may be necessary to use SSH's X forwarding feature, but this change doesn't affect GDM. Contrary to option D, all three XDMCP servers accept remote logins.

14. C. CDE is a commercial desktop environment. KDE, GNOME, and XFce are all open-source packages.

15. C. The `/usr/share/xsessions` directory holds files that define login sessions, including their names and the commands used to launch them. The remaining options are incorrect (options A and D aren't even standard directories).

16. B. In an X startup script such as ˜/.xinitrc, most program calls should terminate in an ampersand (&), which indicates that execution should continue. An exception is the window manager or desktop environment; execution of the script should stop at this point because if it doesn't, the script will terminate before the user is finished (and probably even before the desktop environment can fully start). The ampersand placement is reversed in the sample script, and option B correctly describes the events that will occur when this script is run as part of the startx X startup procedure.

17. B. XDM runs the ˜/.xsession file as part of its login procedure, so option B is correct. The ˜/.bashrc file is one of the startup files for bash, not for XDM. The ˜/.xinitrc file is used when starting X from text mode via startx. The ˜/.Xresources file holds X resources; it's not used to start a desktop environment or window manager.

18. D. Unfortunately, X resource names (as stored in ˜/.Xresources) are not standardized. Thus, you must consult the program's documentation to learn which (if any) of the resources controls the menu font.

19. A. The xhost command controls various aspects of the local X server, including from which remote computers it will accept connections. Option B sets the DISPLAY environment variable, which doesn't directly affect the X server (it does tell X clients which X server to use, though). Option C simply initiates a text-mode remote login session with penguin.example.com. Option D's xaccess is a fictitious program.

20. D. The Secure Shell (SSH) is a remote login tool that's well known for its encryption and thus for its improved security compared to other login methods. By tunneling an X session, you gain the benefits of that security for all the X programs you run, as described in option D. X's bit depth isn't affected by SSH tunneling, contrary to what option A suggests. Likewise, option B's font smoothing isn't affected by SSH tunneling. Although SSH does provide compression features, these features affect data transfer rates, not the size of the image displayed on the screen as option C suggests.

The LPI 102 Exam (99 Weights)

Chapter

6

The Boot Process and Scripts

THE FOLLOWING LINUX PROFESSIONAL INSTITUTE OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **1.105.1: Manage/Query kernel and kernel modules at runtime (weight: 4)**
- ✓ **1.105.2: Reconfigure, build, and install a custom kernel and kernel modules (weight: 3)**
- ✓ **1.106.1 Booting the System (weight: 3)**
- ✓ **1.106.2 Change runlevels and shutdown or reboot system (weight: 3)**
- ✓ **1.109.1 Customize and use the shell environment (weight: 5)**
- ✓ **1.109.2 Customize or write simple scripts (weight: 3)**

The kernel is at the core of every Linux system, and so being able to manage the kernel is an important part of Linux system administration. The kernel is composed of both a main kernel file and supplementary files (known as *kernel modules*) that provide extra drivers and functionality. This chapter begins with a look at managing kernel modules, because this is the easiest way to adjust the kernel's features. A more sophisticated way to customize the kernel is to rebuild it from source code, so that topic is up next. Once the kernel is rebuilt, you must reboot the computer to use the new kernel, and for that you must be able to adjust your boot loader software.

When the computer boots, it runs a key program, `init`, which manages the startup of other processes. Thus, managing `init` provides the means to control the system startup process. A key part of this process is a series of startup shell scripts, so this chapter also describes the Linux shell environment and the shell scripts that they support.

Managing Kernel Modules

Early Linux kernels consisted of a single file, which was loaded into memory by the boot loader. Although modern kernels still use this primary kernel file, Linux has supported kernel modules for several years. Placing most drivers in modules helps keep the size of the main kernel file down, enables the use of a single "generic" kernel on many computers without bloating the kernel's size too much, and gives users control over when and how drivers are loaded. This approach does require learning something about how modules are controlled, though. To begin, you must know how to obtain information on your kernel version and the modules that are currently loaded. Loading modules is also important for their use, and this task can be accomplished both by configuring boot files for modules you want to be used frequently and by using one-off commands. Similarly, unloading kernel modules can be important in some cases. Finally, some tools can help you maintain your kernel modules, such as changing default options and updating system information about available modules.

Obtaining Information about the Kernel and Its Modules

Before tweaking your kernel modules, you should know something about the kernel and the already installed modules. This information can be helpful because it can inform your decision of whether or not loading a new module is necessary, what modules are being used by other modules, and so on.

Learning about the Kernel

You can obtain the most important information about the kernel via the `uname` command:

```
$ uname -a
Linux nessus 2.6.6 #12 Sun May 30 23:22:28 EDT 2004 x86_64
➥AMD Athlon(tm) 64 Processor 3000+ AuthenticAMD GNU/Linux
```

This program provides several types of information, and you can tell it what information to provide with various options:

All information   The `-a` or `--all` option, as used in this example, provides all available information. This is usually a safe option to use, although if you know precisely what information you need, you can create a less cluttered display by using a more specific option or set of options.

Kernel name   The `-s` or `--kernel-name` option produces the name of the kernel, which is normally `Linux`.

Network hostname   The `-n` or `--nodename` displays the network node's hostname (`nessus` in the preceding example). This name is the one that's locally configured; depending on network Domain Name System (DNS) server configurations and client options, the computer might or might not actually be reachable by this name.

Kernel release number   The `-r` or `--kernel-release` option displays the kernel release number (`2.6.6` in the preceding example). This information is often very important because it can influence what driver modules may be used with the kernel.

Kernel version information   The `-v` or `--kernel-version` option displays additional kernel version information. This information does *not* include the kernel version number, though, as displayed by the `-r` option. It does include the kernel build date and time (`#12 Sun May 30 23:22:28 EDT 2004` in the preceding example).

Machine architecture code   The `-m` or `--machine` option displays a code for the CPU—`x86_64` in the preceding example, referring to an AMD64 (aka *x*86-64 or EM64T) CPU.

Machine CPU description   The `-p` or `--processor` option displays a description of the CPU, as in `AMD Athlon(tm) 64 Processor 3000+` in the preceding example. This information is often quite helpful if you need to evaluate the CPU's speed on a system with which you're unfamiliar.

Hardware platform   The `-i` or `--hardware-platform` option displays an identification string provided by the CPU—`AuthenticAMD` in the preceding example.

OS information   The `-o` or `--operating-system` option displays the name of the OS (`GNU/Linux` in the preceding example). This is distinct from the kernel name (the `-k` option) because it refers to the OS as a whole, not just the kernel.

Program information   The `--help` option displays a summary of the program's options, while `--version` displays the `uname` version number.

> **NOTE**   Some of these options may seem odd. After all, you probably know perfectly well that you're running Linux, so why use a -k or -o option? The uname tool is available on non-Linux Unix-like OSs, though, and so it's a handy way for cross-platform scripts to learn something about the OS on which they're running. The scripts can then adjust themselves to work correctly on Linux, FreeBSD, Solaris, Mac OS X, or other environments. If you write such a script, be aware that some OSs support only the long forms of the options (say, --operating-system rather than -o), so you should use the long form in scripts that might run on non-Linux systems.

For your own information, the kernel release information is likely to be the most important. Some third-party kernel modules come in precompiled forms that work only with certain kernels, so if your kernel version doesn't match, you may need to rebuild the right version, as described later, in "Creating a Custom Kernel." The architecture and CPU data can also be important if you don't already know this information. Some programs work only with certain CPUs, and if you need to evaluate the speed of a system you've not used before, this can provide you with some important basic data. Precompiled kernel modules also typically work only with one type of CPU. For instance, you couldn't load a binary module for an *x*86 CPU on an AMD64 kernel such as the one that produced the preceding example output.

Learning about the Kernel Modules

You can learn about the modules that are currently loaded on your system by using lsmod, which takes no options and produces output like this:

```
$ lsmod
Module                  Size  Used by
isofs                  35820  0
zlib_inflate           21888  1 isofs
floppy                 65200  0
nls_iso8859_1           5568  1
nls_cp437               7296  1
vfat                   15680  1
fat                    49536  1 vfat
sr_mod                 19236  0
ide_cd                 42848  0
cdrom                  39080  2 sr_mod,ide_cd
```

> **NOTE**   This output has been edited for brevity. Although outputs this short are possible with certain configurations, they're rare.

The most important column in this output is the first one, labeled `Module`; this column specifies the names of all the modules that are currently loaded. You can learn more about these modules with `modinfo`, as described shortly, but sometimes their purpose is fairly obvious. For instance, the `floppy` module provides access to the floppy disk drive.

The `Used by` column of the `lsmod` output describes what's using the module. All the entries have a number, which indicates the number of other modules or processes that are using the module. For instance, in the preceding example, the `isofs` module (used to access CD-ROM filesystems) is not currently in use, as revealed by its `0` value, but the `vfat` module (used to read VFAT hard disk partitions and floppies) is being used, as shown by it value of `1`. If one of the modules is being used by another module, the using module's name appears in the `Used by` column. For instance, the `isofs` module relies on the `zlib_inflate` module, so the latter module's `Used by` column includes the `isofs` module name. This information can be useful when managing modules. For instance, if your system produced the preceding output, you couldn't directly remove the `zlib_inflate` module because it's being used by the `isofs` module, but you could remove the `isofs` module, and after doing so you could remove the `zlib_inflate` module. (Both modules would need to be added back to read most CD-ROMs, though.)

> The `lsmod` command only displays information on kernel modules, not on drivers that are compiled directly into the Linux kernel. For this reason, a module might need to be loaded on one system but not on another to use the same hardware because the second system might compile the relevant driver directly into the kernel.

You can learn still more about kernel modules with the help of the `modinfo` command. Normally, you type this command followed by the name of the module in which you're interested:

```
$ modinfo vfat
author:        Gordon Chaffee
description:   VFAT filesystem support
license:       GPL
depends:       fat
vermagic:      2.6.6 preempt gcc-3.4
```

The information returned usually includes the author's name, a brief description of the module's function, its license name, the names of any modules upon which it depends, and some kernel and compiler version information. The exact information returned depends on the module, though; some omit some of these fields or add others. If you're only interested in one field, you can specify it with the `-F` *fieldname* option, as in **`modinfo -F description vfat`** to obtain the description for the `vfat` module.

The `modinfo` utility is most useful for learning a bit about modules you've seen in `lsmod` output that you can't readily identify. Unfortunately, many modules lack the helpful description field, so in practice, `modinfo` is often less helpful than it might be.

Loading Kernel Modules

Linux enables you to load kernel modules with two programs: insmod and modprobe. The insmod program inserts a single module into the kernel. This process requires that any modules upon which the module you're loading relies are already loaded. The modprobe program, by contrast, automatically loads any depended-on modules and so is generally the preferred way to do the job.

> **NOTE** In practice, you may not need to use insmod or modprobe to load modules because Linux can load them automatically. This ability relies on the kernel's module auto-loader feature, which must be compiled into the kernel, and on various configuration files, which are also required for modprobe and some other tools. Using insmod and modprobe can be useful for testing new modules or for working around problems with the auto-loader, though.

In practice, insmod is a fairly straightforward program to use; you type it followed by the module filename:

```
# insmod /lib/modules/2.6.6/kernel/drivers/block/floppy.ko
```

This command loads the floppy.ko module, which you must specify by filename. Modules have module names, too, which are usually the same as the filename but without the extension, as in floppy for the floppy.ko file. Unfortunately, insmod requires the full module name.

You can pass additional module options to the module by adding them to the command line. Module options are highly module-specific, so you must consult the documentation for the module to learn what to pass. Examples include options to tell an RS-232 serial port driver what interrupt to use to access the hardware or to tell a video card framebuffer driver what screen resolution to use.

Some modules depend on other modules. In these cases, if you attempt to load a module that depends on others and those other modules aren't loaded, insmod will fail. When this happens, you must either track down and manually load the depended-upon modules or use modprobe. In the simplest case, you can use modprobe just as you use insmod, by passing it a module name:

```
# modprobe floppy
```

As with insmod, you can add kernel options to the end of the command line. Unlike insmod, you specify a module by its module name rather than its module filename when you use modprobe. Generally speaking, this helps make modprobe easier to use, as does the fact that modprobe automatically loads dependencies. This greater convenience means that modprobe relies on configuration files, as described in "Maintaining Kernel Modules." It also means that you can use options (placed between the command name and the module name) to modify modprobe's behavior:

Be verbose   The -v or --verbose option tells modprobe to display extra information on its operations. Typically, this includes a summary of every insmod operation it performs.

Change configuration files   The modprobe program uses a configuration file called /etc/modprobe.conf. You can change the file by passing a new file with the -C *filename* option, as in **modprobe -C /etc/mymodprobe.conf floppy**.

Perform a dry run   The -n or --dry-run option causes modprobe to perform checks and all other operations *except* for the actual module insertions. You might use this option in conjunction with -v to see what modprobe would do without actually loading the module. This might be helpful in debugging, particularly if inserting the module is having some detrimental effect, such as disabling disk access.

Remove modules   The -r or --remove option reverses modprobe's usual effect; it causes the program to remove the specified module and any upon which it depends. (Depended-upon modules are *not* removed if they're in use, though.)

Force loading   The -f or --force option tells modprobe to force the module loading even if the kernel version doesn't match what the module expects. This action is potentially dangerous, but it's occasionally required when using third-party binary-only modules.

Show dependencies   The --show-depends option shows all the modules upon which the specified module depends. This option doesn't install any of the modules; it's purely informative in nature.

Show available modules   The -l or --list option displays a list of available options whose names match the wildcard you specify. For instance, typing **modprobe -l v*** displays all modules whose names begin with v. If you provide no wildcard, modprobe displays all available modules. Like --show-depends, this option doesn't cause any modules to be loaded.

> This list of options is incomplete. The others are relatively obscure, so you're not likely to need them very often. Consult the modprobe man page for more information.

Removing Kernel Modules

In most cases, you can leave modules loaded indefinitely; the only harm that a module does when it's loaded but not used is to consume a small amount of memory. (The lsmod program shows how much memory each module consumes.) Sometimes, though, you might want to remove a loaded module. Reasons include reclaiming that tiny amount of memory, unloading an old module so that you can load an updated replacement module, and removing a module that you suspect is unreliable.

   The actual work of unloading a kernel module is done by the rmmod command, which is something of the opposite of insmod. The rmmod command takes a module name as an option, though, rather than a module filename:

```
# rmmod floppy
```

This example command unloads the `floppy` module. You can modify the behavior of `rmmod` in various ways:

Be verbose   Passing the `-v` or `--verbose` option causes `rmmod` to display some extra information about what it's doing. This might be helpful if you're troubleshooting a problem.

Force removal   The `-f` or `--force` option forces module removal even if the module is marked as being in use. Naturally, this is a very dangerous option, but it's helpful sometimes if a module is misbehaving in some way that's even more dangerous. This option has no effect unless the `CONFIG_MODULE_FORCE_UNLOAD` kernel option is enabled.

Wait until unused   The `-w` or `--wait` option causes `rmmod` to wait for the module to become unused, rather than return an error message, if the module is in use. Once the module is no longer being used (say, after a floppy disk is unmounted if you try to remove the `floppy` module), `rmmod` then unloads the module and returns. Until then, `rmmod` doesn't return, making it look like it's not doing anything.

A few more `rmmod` options exist; consult the `rmmod man` page for details.

Like `insmod`, `rmmod` operates on a single module. If you try to unload a module that's depended upon by other modules or is in use, `rmmod` will return an error message. (The `-w` option modifies this behavior, as just described.) If the module is depended upon by other modules, those modules are listed, so you can decide whether to unload them. If you want to unload an entire *module stack*—that is, a module and all those upon which it depends— you can use the `modprobe` command and its `-r` option, as described earlier in "Loading Kernel Modules."

Maintaining Kernel Modules

Tools such as `lsmod`, `modprobe`, and `rmmod` are very useful for managing kernel modules. These tools rely on configuration files, though, and knowing how to maintain these files is important for keeping your modules operating smoothly. Most of these files, and the tools that help modify them, need only be touched after you add or remove a module to your collection. Sometimes, though, you might want to change the way a module operates by passing it kernel options; this can be done even if you've not recompiled, added, replaced, or removed a module.

Kernel Module Maintenance Tools and Files

Two files help manage some important kernel module features:

Module dependencies   Module dependencies are stored in the `modules.dep` file, which resides in your main modules directory, `/lib/modules/version`, where `version` is the kernel version number. (This number sometimes includes distribution-specific codes.) You don't normally edit this file directly; instead, you use `depmod` to work on it. Typing **depmod**, with no options, as `root` will rebuild the `modules.dep` file for the modules in the current kernel's modules directory. This action also occurs automatically when you install kernel modules, as described later in "Putting Everything in Its Place."

Module configuration   The main module configuration file is `/etc/modules.conf`. This file holds module aliases (that is, alternate names for modules), module options, and more. This file's format is surprisingly complex, but most changes can be relatively simple, as described shortly in "Passing Options to Kernel Modules." Very old distributions called this file `/etc/conf.modules`, but this name has fallen out of favor.

> **WARNING**   Some distributions, such as Debian and Gentoo, use a tool called `modules-update` to dynamically build the `/etc/modules.conf` file from files in the `/etc/modules.d` directory. If you use such a distribution and want to change `/etc/modules.conf`, you should do so by editing the appropriate file in `/etc/modules.d` or by creating a new file there and then typing **`modules-update`**.

Passing Options to Kernel Modules

You can pass options to kernel modules via `insmod` or `modprobe`; however, Linux will usually load a module automatically when it determines that you're trying to use a device. When this happens, you can't manually pass options to the kernel module. Instead, you must edit `/etc/modules.conf` (or a file in `/etc/modules.d`) to tell the system about the options you want to pass. To do so, you must add an `options` line, such as this:

```
options sisfb mode=1280x1024 rate=75
```

This line specifies options for the `sisfb` module, which is the SiS framebuffer driver—that is, it handles unusual text-based video modes for certain SiS video chipsets. (Framebuffer drivers can also be used by X via the X framebuffer driver, but X more often drives the video hardware more directly.) This example passes two options to the `sisfb` module: `mode=1280x1024` and `rate=75`. These options tell the driver to run at a resolution of 1280×1024 with a refresh rate of 75Hz.

Unfortunately, driver options are very driver-centric. For instance, if you were to use the `vesafb` video driver (which works with many VESA-compatible video cards) rather than the `sisfb` driver, you wouldn't use `mode=` and `rate=` options; instead, you'd use a `vga=` option, which takes a numeric code to set the video mode. Your best bet to learn about the options you might want to use is to consult the driver's documentation. For standard kernel drivers, peruse the `/usr/src/linux/Documentation` directory and its subdirectories. If you need to pass options to a driver that didn't ship with a standard kernel, consult the documentation that came with it.

Once you make changes to this file, you should type **depmod**, and if the module whose behavior you want to affect is loaded, unload it. When you reload it, the new options should take effect. A few modules, such as framebuffer video drivers, cannot be easily removed once loaded, so you might need to reboot the computer to see your changes take effect.

Creating a Custom Kernel

Because Linux is an open-source OS, it's possible to completely rebuild the kernel from scratch. Many new Linux administrators are intimidated by this task; it sounds complex and potentially dangerous. If you take a few precautions, though, rebuilding a kernel isn't particularly risky (see "Putting Everything in Its Place"). Picking the right components *is* somewhat complex, though; to do a good job, you need to know a lot about your hardware—the type of CPU you've got, the brand of hard disk controller, and so on. Still, if you make conservative choices, you can be reasonably sure the new kernel will work correctly, and you should have the old one as a fallback.

To build a custom kernel, you must first understand something about how Linux kernels are numbered, lest you download the wrong one. The next tasks are to obtain the kernel source code, configure it, compile it, and store files in their places. Of these tasks, configuring the kernel is definitely the hardest part. Once all of this is done, you must add the kernel to your boot loader, as described later in "Configuring a Boot Loader."

Kernel Version Numbering

Because the kernel is arguably the most important software component in a Linux computer, it's particularly important that you understand how Linux kernel version numbers work. There's a definite system to the Linux kernel version numbers, and understanding it can help you determine when you might want to upgrade the kernel. Unfortunately, the kernel version numbering system has changed with the 2.6.*x* kernel tree, so different rules apply to older and newer kernels.

> Don't confuse the Linux *kernel* version number with the Linux *distribution* version number. The two are unrelated. For instance, Fedora Linux 3.0 (a distribution) ships with kernel 2.6.9. Some new Linux administrators merge the two and say they're running (for instance) "Linux 3.0." There is no such thing—at least, not currently. "Linux 2.6.9" unambiguously refers to the kernel version 2.6.9, but when referring to a distribution, you *must* specify its name.

Pre-2.6.*x* Kernel Version Numbers

Linux kernel versions up to the 2.6.*x* series all have three numbers, separated by dots (.):

- The first number is the *major version number*. In 2005, this number is 2. A change in the major version number typically indicates a fairly substantial change in the way the kernel works.

- The second number carries special meaning in pre-2.6.*x* kernels. When this number is even, it indicates a *stable kernel* or *release kernel*—one that the kernel developers believe to be fairly bug free and ready for use by the public at large. An odd second number indicates a

development kernel. Such a kernel is experimental—it contains major new drivers, major rewrites of existing code, and so on. You should use a development kernel only if you're desperate for some feature it provides—and even then, you can often find a backport (the transfer of newer drivers to older kernels) of such features to a stable kernel.

- The final number indicates a minor upgrade. In the case of stable kernels, these numbers typically indicate small bug fixes and occasionally the addition of an important but well-tested new driver. In the case of development drivers, the final number indicates progression in adding features, fixing bugs, and (being realistic) adding new bugs along with the new features.

Traditionally, when a new stable kernel was released, the former development kernel was abandoned and work was begun on a new line—2.5.*x* after the release of 2.4.0, for instance.

Major Linux distributions always ship with stable kernels. As a general rule, it's best to use a kernel in the same series that the distribution uses. Changes to the second value of the kernel number tend to introduce changes to kernel interfaces required by some low-level utilities, modifications to filesystem formats, new driver names, and the like. Although most distributions continue to function with such an upgraded kernel, there may be a few glitches, such as drivers that don't load automatically.

Within a single series, upgrades are usually safe. The most common reason to upgrade is if a new version fixes a bug that's been found in an older version. Another reason to upgrade is if a new version adds a driver.

2.6.*x* and Later Kernel Version Numbers

The traditional kernel version numbering scheme just described has its problems. For instance, the delay between new stable kernel branches (2.2.*x* to 2.4.*x*) has often been quite long. As a result, important new features, such as support for USB devices, have often been *backported* from the development branch to the stable branch—that is, the new code has been adapted and merged into the older kernel. Such backports have often been included in distributions' kernels, but less often included in the official kernel release. The result has been a lot of variability in precisely what features exist in a compiled stable kernel, depending on from where it was obtained.

As a result, with the 2.6.*x* kernels, a new numbering system has been adopted, with the goal of speeding up the uptake of new features in stable kernels. This new system involves four numbers—*a.b.c.d*, as in 2.6.11.10. In this system, the first two numbers (*a* and *b*) have the same meaning as in the traditional system, except that the even or odd status of the second number (*b*) is unimportant. An increment to the third number (*c*) represents significant changes to the kernel, as maintained by Linus Torvalds himself. Increments within this series (to the fourth number, *d*) represent "trivial" patches. Despite the word *trivial* in the official discussions of these matters, these changes can be quite important, such as bug fixes and security patches. The word *trivial* refers more to the size and scope of the change from a programming perspective than to its importance to end users.

Under this new system, the stable/unstable distinction is essentially lost. This fact may make it harder for administrators to pick a kernel that will be suitable for a production system; however, the new naming scheme is new enough at press time that its long-term practical effects are still uncertain.

Obtaining the Kernel

Before you can compile your own kernel, you must obtain the kernel source code. One good source for the "official" kernel is `http://www.kernel.org`. This site maintains a set of kernels as released by the Linux kernel developers, without any third-party add-ons. If this is the first time you've obtained a kernel, or if you haven't done so in a while, follow the links to the HTTP or FTP repository (either one works), to the Linux Repository, to the kernel, and to the kernel version (such as v2.6 for the 2.6.*x*.*y* kernels). You'll see a directory filled with files, most with names of the form `linux-`*version*`.tar.*` and `patch-`*version*`.*`, where *version* is the version number and `*` is an extension indicating the compression tool used. Some filenames end in `.sign`, indicating digital signatures verifying the authenticity of the files with similar names but without the `.sign` extension. Download an appropriate `linux-`*version*`.tar.bz2` file. The `patch-`*version*`.*` files contain patches, which you can use to upgrade the immediately preceding kernel source code if you've already got it. The files with `.bz2` extensions are compressed with `bzip2`, which is more efficient than the older `gzip` compression, indicated by a `.gz` extension.

> **NOTE**   Complete Linux kernel files are quite large. The 2.6.11.10 kernel that's current as I write is 35MB, even with `bzip2` compression. (It's 44MB with `gzip` compression.) If your network connection is slow, you could spend several minutes downloading the file. Also, be sure that you've got enough free disk space to hold the kernel.

Once you've downloaded the kernel, you may also want to download the matching `.sign` file. You can then verify the authenticity of the main kernel file. To do so, you must first import the public key used by the kernel maintainers:

```
$ gpg --keyserver wwwkeys.pgp.net --recv-keys 0x517D0F0E
```

> **NOTE**   If you've not used gpg before, the utility will deliver messages relating to the creation of its directories and key rings. Whether or not you've run it before, it will produce several lines out output summarizing its actions.

Be sure to type this command *exactly* as it's shown here. The key number (`0x517D0F0E`) is particularly important because it identifies the key that's to be used. Once this is done, you can verify the file's authenticity:

```
$ gpg --verify linux-2.6.11.10.tar.bz2.sign linux-2.6.11.10.tar.bz2
gpg: Signature made Mon May 16 14:28:22 2005 EDTusing DSA key ID
➥517D0F0E
gpg: Good signature from "Linux Kernel Archives Verification Key
➥<ftpadmin@kernel.org>"
```

These lines confirm that the file is valid. You'll also probably see a warning to the effect that the key isn't certified with a trusted signature. This warning relates to the fact that the GNU Privacy Guard (GPG) program can't verify the identity of the public key site (`wwwkeys.pgp.net`). Unless a miscreant has managed to compromise both that site and the Linux kernel site, though, this shouldn't be a problem. If you're paranoid, you can set up a trust path to this site, but this task is beyond the scope of this book.

Another possible source for kernel source code is your distribution. Distribution-provided kernels frequently include patches to the kernel—non-standard drivers and other tweaks that the distribution's maintainers have collected from various sources. These tweaks can sometimes help a kernel work well, but they also make the distribution's kernels slightly non-standard, which can complicate troubleshooting if a problem is related to such changes. Perhaps the biggest advantage of using a kernel from the distribution maintainer is that its default configuration will match that of the distribution's original kernel, which presumably works well enough to get your system running. This can simplify the configuration process.

Now that you've obtained the kernel and verified its validity, you can extract the files and prepare to configure it. Traditionally, the Linux kernel resides in `/usr/src/linux`, but this location is usually a symbolic link to another directory, such as `/usr/src/linux-2.6.11.10`. (Some versions of Fedora and Red Hat include the beginnings of a version number in their main links, as in `/usr/src/linux-2.6` rather than `/usr/src/linux`. For brevity, I won't mention this variation again in this chapter.) You can extract the kernel file by changing into the `/usr/src` directory and using `tar`:

```
$ cd /usr/src
$ tar xvjf ~/linux-2.6.11.10.tar.bz2
$ rm linux
$ ln -s linux-2.6.11.10 linux
```

> **NOTE** Chapter 2, "Managing Software," describes the use of `tar` in more detail.

As presented, these commands will work only if you have write permission to `/usr/src`. This directory is usually owned by `root`, and only `root` may write to it, so you'll either need to do this job as `root` or change the permissions on `/usr/src` so that you can write to it as an ordinary user. Alternatively, you could extract the kernel source code to another directory (say, `/home/yourdir/kernels`) and create a symbolic link from `/usr/src/linux` to the kernel directory in `/home/yourdir/kernels`. Because most of the kernel compilation process doesn't require `root` privileges, it's a bit safer to do as much of it as possible as an ordinary user.

If the existing `/usr/src/linux` directory is a true directory and not a symbolic link, you should rename it rather than remove it with `rm`. Using `rm`, as just shown, is appropriate if `/usr/src/linux` is a symbolic link.

At this point, your kernel should be extracted and ready to configure and compile. Check the `/usr/src/linux` directory. It should contain about two dozen files and directories, such as `kernel`, `arch`, `drivers`, and of course `README`. Naturally, you should read the `README` file, or at least skim it. This file contains information on compiling the kernel, and it might have important notes about changes from the procedures described here.

Configuring the Kernel

The most time-consuming part of kernel compilation is likely to be configuring it. To do so, you must first understand the basic tools you use to compile the kernel. With that knowledge in hand, you can begin examining the kernel configuration options, which are quite numerous.

Using Kernel Configuration Tools

To begin configuring the kernel, change into the kernel source directory and type a `make` command to configure the kernel in any of several ways:

Adapt old configuration   The first option is to adapt an old configuration. To do so, you must first copy the kernel configuration file (`.config`) from an existing kernel directory to the new one. The existing directory could be your distribution's older kernel source or a kernel you've previously compiled. Either way, type **`make oldconfig`** and the system will run through a text-mode configuration (described next), but it will only ask you about new or changed options. This approach is particularly handy if you've already compiled an earlier kernel or if you want to duplicate your distribution's configuration.

Text-mode configuration   The basic configuration is done by typing **`make config`**. The result is a series of questions asking you about how to configure every option in the kernel, omitting only those options that are moot because of earlier answers. (For instance, you won't be asked about SCSI card drivers if you tell the system you don't want SCSI support.) If you make a mistake, you can't go back and correct it with this method; it simply plows through the questions with no way to change the order or revisit an option. This approach is very tedious, and it's best used as a last-ditch method.

Text-mode menu configuration   Typing **`make menuconfig`** produces a menu-based text-mode configuration tool. You can scroll through the menu items, enter those you want to enter, ignore those you want to ignore, and go forward or back to configure components in any order you want. This is a flexible approach to configuration that works in text-mode logins or in `xterm` or similar command prompt windows within X.

GUI configuration   If you're running X, you can type **`make xconfig`** to get an X-based configuration window, as shown in Figure 6.1. This system is functionally identical to typing **`make menuconfig`**, but it uses mouse clicks and the GUI display.

Linux kernel options are divided into several layers of categories and subcategories. In Figure 6.1, these categories are displayed in the pane on the left of the window. Click a category to see the specific options it contains in the top-right pane. Click one of these options to see a description. The box to the left of the option name is empty if the option won't be compiled, holds a check if the option is to be compiled into the main kernel file, or contains a dot if the option is to be compiled as a kernel module. Clicking within the box changes the build status between these three options. (Some options support only modular or only in-kernel compilation or change their available choices depending on other choices you've made.) If you're using a text-mode tool, pressing the spacebar selects how the option is to be compiled; an asterisk (*) stands for in-kernel compilation and an M stands for modular compilation.

The most convenient method of Linux kernel configuration uses a GUI to
enable you to select kernel options.
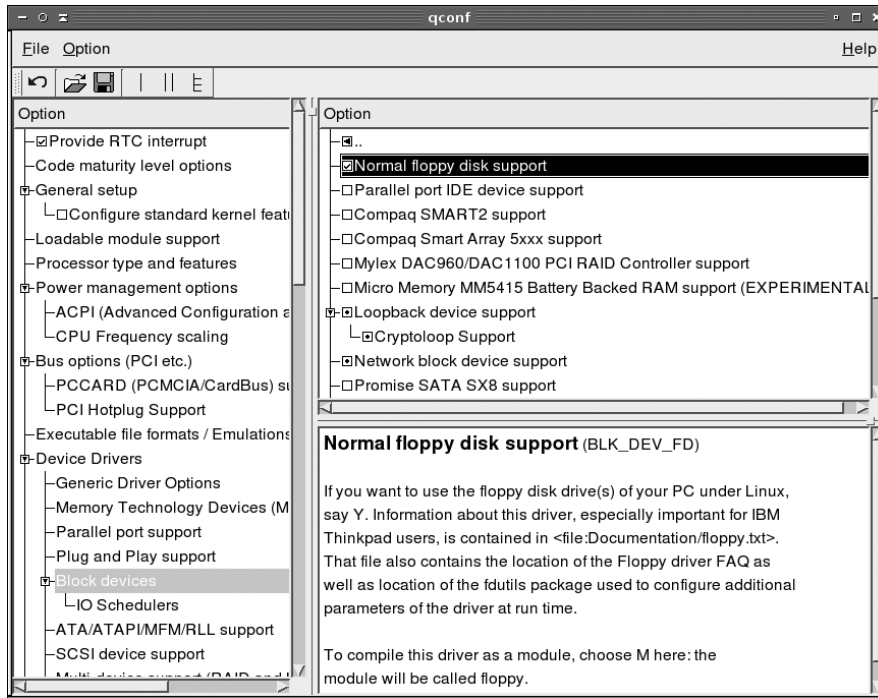
Real World Scenario

Picking the Kernel Configuration Method

In practice, using make xconfig is usually the best approach to configuring the kernel if you're
running X. The X-based kernel configuration interface is the easiest to use if you're comfortable
with an X-based environment. Sometimes, though, this method doesn't work because of miss-
ing development libraries. If you run into this problem, you can either track down whatever
libraries are reported as missing and install them or fall back on a text-mode configuration
method. Generally speaking, make menuconfig is not too tedious, but make config is very pain-
ful, so you should use it only as a last resort.

Occasionally, make menuconfig fails when run from an xterm or other GUI command window. If this
happens, try expanding the size of the window—they're sometimes resized to be slightly smaller
than make menuconfig requires, so increasing the window size works around this problem.

Picking Kernel Options

Linux kernel options are far too numerous to describe them all. Many options are highly hardware-specific—they're drivers for particular disk controllers, sound cards, and so on, or they're options related to supported CPUs or other architectural features. In fact, the precise options you see depend on your CPU, as detected by the configuration routines.

As a general rule, it's best to start with a known working configuration, in the form of a .config file copied from the kernel directory of the kernel you're using. (If necessary, install your distribution's kernel source package to get this file.) You can then use **make oldconfig**, as just described, to update the file for the new kernel. If you like, you can then run **make menuconfig** or **make xconfig** to peruse and adjust both new and old kernel options. If you don't understand an option, leave it alone or set it at the default setting suggested in the option description. Options that deserve special attention include:

CPU settings   The Processor Type and Features area contains options relating to the CPU. For best performance, set your CPU type correctly. On 32-bit $x86$ systems, many distributions ship with simple 386 optimizations, which work on all systems but don't take best advantage of modern CPUs. Setting your CPU type correctly will enable optimizations that can slightly increase system performance.

Hard disk controllers   Most systems use Advanced Technology Attachment (ATA) hard disks, which interface via an ATA controller. Linux ATA drivers are enabled in the ATA/ATAPI/MFM/RLL Support area. A generic ATA driver supports all ATA controllers in a very slow legacy mode, but to get good performance, you must activate the driver for your ATA controller. It's best to compile this driver into the main kernel file rather than compile it as a module, because if it's compiled as a module, you must take extra steps to place that module in an initial RAM disk, which the boot loader can pass to the kernel at boot time. You should also enable the ATA hard disk support (again, ideally in the main kernel file) and probably the CD-ROM option. Some new Serial ATA (SATA) controllers have drivers in the main ATA section, but the trend is to put this in the Serial ATA Support section under the SCSI area, so if you're using an SATA hard drive, you may want to look there. (If you use the SCSI SATA driver, the drive will look like a SCSI drive to Linux.) Likewise, if you're using a real SCSI hard disk, you must activate the SCSI support and support for your SCSI host adapter.

Sound options   Linux provides two main classes of sound drivers: the Open Sound System (OSS) and the Advanced Linux Sound Architecture (ALSA). OSS is the traditional Linux sound system, but it's being phased out; new development is focused on ALSA. Thus, I recommend that you try to get the ALSA drivers working; fall back on the OSS drivers only if you have problems with ALSA or perhaps if you already have a working OSS configuration and want to carry it forward.

Filesystems   Linux supports several filesystems, as described in Chapter 3, "Configuring Hardware." Be sure you compile support for all the filesystems you're using, including Linux native filesystems, non-Linux disk filesystems (such as FAT, NTFS, and HFS), optical disc filesystems (such as ISO-9660 and UDF), and network filesystems (such as NFS and SMB). Most of these can be compiled as modules, but support for the type you are using for your root (/) filesystem should be compiled into the main kernel file. If you don't do this, you must prepare a separate boot RAM disk, which is a nuisance.

Many other kernel options are available, and some may even be critical for your system. For instance, if you use a USB keyboard, you must compile basic USB support and USB human input device (HID) support into the kernel. If you're uncertain about whether to include an option, I recommend you do so, at least as a module. Unused modules do little harm aside from taking some time to compile and consuming a few kilobytes of disk space.

Compiling the Kernel

Actually compiling the kernel is fairly straightforward: Type **make**. The system will begin displaying process messages on the screen as it compiles each file, combines files into modules, and so on. The entire compilation process is likely to take anywhere from a few minutes to over an hour, depending on how many options you've selected and how fast your computer is.

> **NOTE**   With kernels prior to the 2.6.*x* series, you had to type **make modules** to compile modules; the main **make** command only compiled the main kernel file. With the 2.6.*x* kernel, though, typing **make** also compiles the modules.

The result of kernel compilation is a kernel file. On *x*86 systems, this file is /usr/src/linux/arch/i386/boot/bzImage. On other architectures, the file may be called something else, such as vmlinux or vmlinuz, and will be located in another subdirectory of /usr/src/linux/arch, named after the CPU.

> **NOTE**   Distributions often call their stock kernels vmlinuz or vmlinuz-*version*, where *version* is a version number, even on *x*86 systems. Although you can call your kernel this, or anything else you like, this naming is unconventional for locally compiled kernels on the *x*86 architecture.

Compiling the kernel also creates a large number of module files. You needn't be too concerned about them as individual files, though; as described shortly, a simple command will install them all.

Putting Everything in Its Place

Once the kernel is compiled, you must put it and its module files in the appropriate locations. This is the part of the kernel compilation process that requires root privilege, because you'll be copying files to locations to which ordinary users cannot (and should not be allowed to) write.

The first job is to copy the main kernel file to /boot. This can be done with a single cp command:

```
# cp /usr/src/linux/arch/i386/boot/bzImage /boot/bzImage-2.6.11.10
```

This example moves the file and renames it to include the kernel's version number. This practice enables you to keep several kernel files in /boot. (If you want to keep kernels with the same version but different options, try adding additional information to the filename, such as bzImage-2.6.11.10-scsi for a SCSI-enabled kernel.)

> Merely copying the kernel file to /boot isn't enough to use it. You must also configure your boot loader to boot the kernel, as described shortly, in "Configuring a Boot Loader."

Another file you may want to copy to `/boot` is the `System.map` file, which appears in the `/usr/src/linux` directory after you compile the kernel. This file isn't absolutely critical, but it contains kernel driver information that can be helpful in debugging problems.

Installing the kernel module requires typing another `make` command:

```
# make modules_install
```

Note the underscore (`_`) in the `make` target; if you replace it with a space, the command won't work! This command copies all the kernel modules to a subdirectory of `/lib/modules` named after the kernel version, such as `/lib/modules/2.6.11.10` for the 2.6.11.10 kernel.

At this point, the kernel is set up and ready to be booted. To actually boot the system, though, you must reconfigure your boot loader.

Configuring a Boot Loader

In order to be useful, the Linux kernel must be moved from disk to RAM. This process doesn't happen magically; it's the job of the *boot loader,* which is a program that's loaded by the computer's Basic Input/Output System (BIOS) when the computer boots. Specifically, the CPU executes the BIOS, which performs some initial checks and then loads the boot loader from disk. The boot loader can launch other boot loaders (potentially creating a chain of several boot loaders) or load an OS kernel. Thus, to get a kernel you've just compiled to load, you must alter your boot loader's configuration.

Under Linux, two boot loaders are common: the *Linux Loader (LILO)* and the *Grand Unified Boot Loader (GRUB).* GRUB is slowly becoming the standard Linux boot loader, but many distributions still use LILO by default, and most provide at least the option of using LILO. Thus, you should be familiar with at least the basics of using both boot loaders.

> Both LILO and GRUB are primarily boot loaders for the *x*86 and *x*86-64 (aka AMD64 or EM64T) architectures. Other CPUs, such as PowerPC or Alpha, have their own boot loaders. These boot loaders are sometimes modeled after LILO, but sometimes they're not. If you're using Linux on such a system, you should consult your distribution's documentation to learn about its boot loader.

Using LILO as the Boot Loader

LILO was once the default boot loader for Linux on the *x*86 architecture. Although LILO has since been usurped in popularity by GRUB, it is still a small, useful boot loader. To use LILO, you must configure it and install it in your boot sector—the part of the disk that the BIOS (or an earlier boot loader) reads to load the boot loader. Once it's installed, you can reboot the computer and tell LILO which OS or kernel you want to boot.

> The term *LILO* can denote one of two things. The uppercased LILO refers to the boot loader that is installed into your boot sector. The lowercased `lilo` refers to the program that is used to install the LILO boot loader.

Configuring the LILO Boot Loader

LILO is configured through the use of the `/etc/lilo.conf` file. This file is generally broken into two main sections: global and per-image options. (The latter are kept in sections known as stanzas.) Some per-image options are further separated depending on whether they are for a Linux kernel or another OS. To further confound the matter, many of the per-image options may be used at the global level to indicate a default value.

Essential LILO Global Options

Listing 6.1 shows a typical `/etc/lilo.conf` file, including both global options and three stanzas for booting two Linux kernels and a non-Linux OS. Most configuration lines take the form option or `option=value`, but lines that begin with hash marks (#) are comments and are ignored. All but the first line of each stanza are traditionally indented. This practice makes it easy to spot where a new stanza begins.

Listing 6.1: A Typical LILO Configuration File

```
# lilo.conf
#
# Global Options:
#
boot=/dev/hda
prompt
timeout=150
default=fedora
lba32
vga=normal
root=/dev/hda5
read-only
#
# Kernel Options (may have multiple):
```

```
#
image=/boot/vmlinuz-2.6.9
    label=fedora
    initrd=/boot/initrd-2.6.9
    append="mem=512M"
image=/boot/bzImage-2.6.11.10-experimental
    label=debian
    root=/dev/hda6
#
# Other Operating Systems Options (may have multiple):
#
other=/dev/hda2
    label=dos
```

> **NOTE** The system described by Listing 6.1 contains DOS, Fedora Linux, and Debian GNU/Linux. DOS resides on /dev/hda2, Fedora Linux resides on /dev/hda5, and Debian GNU/Linux resides on /dev/hda6. Fedora and Debian share a separate /boot partition, /dev/hda1.

The lilo.conf file supports many options. Most of these are fairly obscure; chances are you'll only need to use a handful of them in order to achieve specific common goals:

Boot loader location The boot= option specifies the name of the device that contains the boot sector (/dev/hda in the case of Listing 6.1). In this example, the first sector (also known as the *master boot record, or MBR*) of the first ATA hard drive is used as the boot sector. In this configuration, LILO functions as the primary boot loader—it's the first one loaded by the BIOS. If you wanted LILO to reside in a partition on /dev/hda, you would give a specific partition identifier, such as /dev/hda1. In this configuration, the BIOS would load a standard *x*86 MBR boot loader, which would then load LILO. This less direct approach is sometimes helpful on multi-boot systems because Windows tends to overwrite the MBR's boot loader when it's installed or upgraded. You can then get LILO back by making LILO's partition bootable using Windows partitioning tools. Placing LILO in the MBR can be simpler in Linux-only installations, though.

Default stanza The default= setting specifies the default kernel or OS that will boot. If this option is omitted, the first image listed in lilo.conf will be used as the default.

Boot prompt The prompt line instructs LILO to issue the boot: prompt and wait for user input. This option is usually desirable, but it can be omitted if you want the system to directly boot a single configuration.

Boot timeout The timeout= setting specifies the amount of time, in tenths of a second, that LILO will wait for keyboard input before booting the default kernel image. You must use the prompt option to enable the timeout. Listing 6.1's setting of 150 indicates a 15-second timeout.

Large disk support   The `lba32` option enables LILO to boot from disks where the kernel image resides on a partition that is past the 1,024th cylinder. This option is almost always desirable, but you might omit it if you're using a very old computer.

Video options   The `vga=` line selects the VGA text mode that is used when booting. Choices include `normal`, `extended`, `ask`, or a number. Unless you're having problems with the video display during the boot process, you should probably leave this alone.

Linux root partition   The `root=` option's value is passed to the Linux kernel to tell it what partition to use as its root (`/`) partition. You can set a default value for this option and override it in individual stanzas, as Listing 6.1 does.

Boot in read-only mode   The `read-only` option indicates that the root file system should be mounted read-only. Usually, the operating system will remount the file system to read-write.

> **TIP**   Want to password protect your booting process? Look into the `password`, `restricted`, `mandatory`, and `bypass` options, which are documented in the man page for `lilo.conf`.

Essential LILO Per-image Options

LILO supports two main types of stanzas: those for Linux kernels and those for other OSs. The `image=` option is used to indicate a Linux kernel and the `other=` option is used to indicate some other OS. Either type of line begins a stanza, and subsequent lines are conventionally indented until the next stanza or the end of the file. You may want to adjust several types of per-image options:

Linux boot image   The `image=` line indicates the Linux kernel file to use when booting. You must pass the complete path to the Linux kernel image file.

Non-Linux boot partition   An `other=` line indicates the partition that contains its own boot loader. When this option is selected, LILO passes control to the boot loader in that partition. DOS, Windows, OS/2, BeOS, FreeBSD, and other OSs can all place their own boot loaders in their partitions, so this tool enables you to pass control to these OSs.

OS label   The `label=` option provides a name for LILO to use. When you boot, you type the name of the label or press the Tab key to get a list of available labels, as described later in "Interacting with LILO."

RAM disk   The `initrd=` line points to an initial RAM disk. This is a small filesystem in a file that the boot loader loads into memory and delivers to the kernel as a RAM-based substitute for a disk drive. Linux distributions frequently use RAM disks like this to store kernel drivers as a way of keeping the kernel size down while still supporting a wide range of drivers. When you build your own kernel, it's usually simpler to build drivers that are necessary to the boot process (such as those for your hard disk controller and the root filesystem) into the main kernel file.

Extra kernel options   You can pass arbitrary options to the kernel with the `append=` option. For instance, Listing 6.1's `fedora` stanza passes the `mem=512M` option to the kernel in this way. (This specific option tells the computer that is has 512MB of RAM. Linux usually detects available RAM correctly, but sometimes this or some other option is necessary.)

In addition, some of the options described in the section "Essential LILO Global Options" can be used in per-image stanzas. In particular, `vga=`, `root=`, and `read-only` are often found in per-image stanzas. If they aren't used in stanzas, the global options apply.

Adding a Kernel to LILO

To add a new kernel to LILO, follow these steps as `root`:

1. Load `/etc/lilo.conf` into your favorite text editor.
2. Copy a working Linux stanza.
3. In the copied stanza, modify the `label=` line to give the copy a new name. The name should be a string of letters, numbers, or both, without spaces.
4. Change the `image=` line to point to the new kernel file.
5. Change any other options that may need changing. For instance, if you've prepared a new RAM disk, change the `initrd=` line to point to it. If your new configuration doesn't use a RAM disk, you can eliminate any `initrd=` line that's present in the stanza.
6. Save your changes and exit from the text editor.
7. At a `root` command prompt, type **lilo**. This command installs LILO in the MBR or boot partition's boot sector. You should see a list of stanza names echo to the screen. Ensure that your new configuration is present.

> **WARNING**   Remember to perform step #7! It's easy to overlook this step, and if you do so, your changes won't take effect. If you forget and reboot before typing **lilo**, you should boot using a working kernel, log in as root, type lilo, and reboot again.

A similar procedure can be used to add a new non-Linux OS; however, you would then copy and modify a working `other=` configuration (or copy the one shown in Listing 6.1).

> **WARNING**   Don't remove old `image=` or `other=` configurations until you are certain that the new images are working. This will allow you to boot to the old kernel image in case of failure. Another alternative is to install LILO with your new configuration onto a temporary device such as a floppy disk or a USB memory stick. To do this, change the `boot=` line to point to the test boot device, such as boot= /dev/fd0 to install LILO to a floppy disk.

You can pass some options to `lilo` to modify what it does:

Specify an alternate configuration file   The `-C` *config-file* option specifies an alternate configuration file to use rather than `/etc/lilo.conf`.

Test the configuration   The `-t` option tests the configuration; it doesn't actually write anything to the boot sector.

Produce verbose output   The `-v` option produces verbose output when `lilo` is run. This is useful with the `-t` option to see how your changes will work out.

Specify a boot device   The `-b` *bootdev* option specifies a boot device, overriding the `boot=` option in `lilo.conf`.

> **WARNING**   It is possible to install a new LILO boot loader and still have it fail when trying to use it. The `lilo` man page describes the errors that may occur. LILO prints the letters L, I, L, and 0 to indicate how it is progressing.

Interacting with LILO

Sometimes you will need to boot with options that you did not place into the `/etc/lilo.conf` file. If you've configured LILO to provide a `boot:` (or sometime `lilo:`) prompt, you may pass extra options to the boot loader on that command line.

One thing that you may wish to do is boot into single user mode. Assuming that the image that you want to use is called `linux`, you would type something like this:

```
boot: linux 1
```

> **NOTE**   Instead of supplying the number 1 for single user mode, you can specify the letter s or S or the word `single`.

It is also possible that your `init` program (usually `/sbin/init`) is corrupt, missing, or incorrectly configured. At times like this, it is possible to specify an alternate `init` program. A common technique is to use a regular shell program like `bash` in place of `init`. You would do this at the `boot:` prompt like this:

```
# boot: linux init=/bin/sh
```

Using GRUB as the Boot Loader

GRUB has taken over as the default boot loader for many Linux distributions because it offers many features that LILO lacks. For example, you do not have to reinstall GRUB after editing its configuration and you have many more interactive options while booting. As with LILO, you must still explicitly add a new kernel to the GRUB configuration if you intend to use that kernel. If GRUB isn't already installed in your boot sector, you must also install it, although this step isn't necessary if GRUB is already working. As with LILO, you can interact with GRUB during the boot process, passing options to control how the system boots.

Configuring the GRUB Boot Loader

The usual location for the GRUB boot loader's configuration file is /boot/grub/menu.lst. Some distributions (such as Fedora, Red Hat, and Gentoo) use the filename grub.conf rather than menu.lst, though. GRUB is able to read its configuration file at boot time, which means that you needn't reinstall the boot loader to the boot sector when you change the configuration file. As with LILO, the GRUB configuration file is broken down into global and per-image sections, each of which has its own options. Before getting into section details, though, you should understand a few GRUB quirks.

GRUB Nomenclature and Quirks

Listing 6.2 shows a sample GRUB configuration file. This file is roughly equivalent to the LILO configuration file shown in Listing 6.1. In particular, it can boot the same OSs using the same kernels—Fedora on /dev/hda5, Debian on /dev/hda6, and DOS on /dev/hda2. Fedora and Debian share a /boot partition (/dev/hda1), on which the GRUB configuration resides.

Listing 6.2: A Sample GRUB Configuration File

```
# grub.conf/menu.lst
#
# Global Options:
#
default=0
timeout=15
splashimage=/grub/bootimage.xpm.gz
#
# Kernel Image Options:
#
title Fedora (2.6.9)
   root (hd0,0)
   kernel /vmlinuz-2.6.9 ro root=/dev/hda5 mem=512M
   initrd /initrd-2.6.9
title Debian (2.6.11.10-experimental)
   root (hd0,0)
   kernel (hd0,0)/bzImage-2.6.11.10-experimental ro root=/dev/hda6
#
# Other operating systems
#
title DOS
   rootnoverify (hd0,1)
   chainloader +1
```

The GRUB boot loader does not refer to disk drives by device filename the way that Linux does. GRUB numbers drives so that instead of `/dev/hda`, GRUB uses (`hd0`). Similarly, `/dev/hdb` would be (`hd1`). GRUB doesn't distinguish between ATA and SCSI drives, so on a SCSI-only system, the first SCSI drive will be (`hd0`). On a mixed ATA-and-SCSI system, ATA drives normally receive the lower numbers, although this isn't always the case.

Additionally, GRUB numbers partitions on a drive starting at 0 instead of the 1 that is used by Linux. GRUB separates partition numbers from drive numbers with a comma, as in (`hd0,0`) for the first partition on the first disk (normally Linux's `/dev/hda1`) or (`hd0,4`) for the first logical partition on the first disk (normally Linux's `/dev/hda5`). Floppy devices are referred to as (`fd0`), or conceivably (`fd1`) or above if you have more than one floppy drive. Floppy disks aren't partitioned, so they don't receive partition numbers.

GRUB defines its own root partition, which can be different from the Linux root partition. GRUB's root partition is the partition in which GRUB's configuration file (`menu.lst` or `grub.conf`) resides. Because this file is normally in Linux's `/boot/grub/` directory, the GRUB root directory will be the same as Linux's root directory if you do *not* use a separate `/boot` or `/boot/grub` partition. If you split off `/boot` into its own partition, though, as is fairly common, GRUB's root partition will be the same as Linux's `/boot` partition. You must keep this difference in mind when referring to files within the GRUB configuration directory.

Essential Global GRUB Options

GRUB's global section precedes its per-image configurations. Typically, you'll find fewer options in this global section than in a LILO global configuration:

Default OS   The `default=` option tells GRUB which OS to boot. Listing 6.2's `default=0` causes the first listed OS to be booted (remember GRUB indexes from 0). If you wanted to boot the second listed operating system, use `default=1`, and so on through all your OSs.

Timeout   The `timeout=` option defines how long, in seconds, to wait for user input before booting the default operating system. Note that GRUB measures its timeout period in seconds, whereas LILO uses tenths of a second.

Background graphic   The `splashimage=` line points to a graphics file that's displayed as the background for the boot process. This line is optional, but most Linux distributions point to an image to spruce up the boot menu. The filename reference is relative to the GRUB root partition, so if `/boot` is on a separate partition, that portion of the path is omitted. Alternatively, the path may begin with a GRUB device specification, such as (`hd0,5`) to refer to a file on that partition.

Essential GRUB Per-Image Options

GRUB's per-image options are typically indented after the first line, much as are LILO's, but this is a convention, not a requirement of the file format. The options begin with an identification and continue with options that tell GRUB how to handle the image:

Title   The `title` line begins a per-image stanza and specifies the label to display when the boot loader runs. Unlike LILO's `label` option, the GRUB `title` can accept spaces and is conventionally much more descriptive, as shown in Listing 6.2.

GRUB root   The `root` option specifies the location of GRUB's root partition. This would be the `/boot` partition if a separate one exists; otherwise it's usually the Linux root (/) partition. (GRUB *can* reside on a FAT partition, on a floppy disk, or on certain other OSs' partitions, though, so GRUB's root could conceivably be somewhere more exotic.)

Kernel specification   The `kernel` setting describes the location of the Linux kernel as well as any kernel options that are to be passed to it. Paths are relative to GRUB's root partition. As an alternative, you can specify devices using GRUB's syntax, such as `kernel (hd0,5)/vmlinuz ro root=/dev/hda5`. Note that you pass most kernel options on this line. LILO splits off kernel options on separate lines, such as the `append=` line, the `root=` line, and the `read-only` option. In GRUB, you incorporate these options onto the kernel line. The `ro` option tells the kernel to mount its root filesystem read-only (it's later remounted read/write), and the `root=` option specifies the *Linux* root filesystem. Because these options are being passed to the kernel, they use Linux-style device identifiers, when necessary, unlike other options in the GRUB configuration file.

Initial RAM disk   Use the `initrd` option to specify an initial RAM disk, much like the option of the same name in LILO.

Non-Linux root   The `rootnoverify` option is similar to the `root` option except that GRUB will not try to access files on this partition. It's used to specify a boot partition for OSs for which GRUB can't directly load a kernel, such as DOS and Windows.

Chain loading   The `chainloader` option tells GRUB to pass control to another boot loader. Typically, it's passed a `+1` option to load the first sector of the root partition (usually specified with `rootnoverify`) and to hand over execution to this secondary boot loader.

To add a kernel to GRUB, follow these steps:

1. As `root`, load the `menu.lst` or `grub.conf` file into a text editor.
2. Copy a working configuration for a Linux kernel.
3. Modify the `title` line to give your new configuration a unique name.
4. Modify the `kernel` line to point to the new kernel. If you need to change any kernel options, do so.
5. If you're adding, deleting, or changing a RAM disk, make appropriate changes to the `initrd` line.
6. If desired, change the global `default` line to point to the new kernel.
7. Save your changes and exit from the text editor.

At this point, GRUB is configured to boot your new kernel. When you reboot, you should see it appear in your menu and you should be able to boot it. If you have problems, boot a working configuration to debug the issue.

> **WARNING**   Don't eliminate a working configuration for an old kernel until you've determined that your new kernel works correctly.

Installing the GRUB Boot Loader

Installation of GRUB is a little different than for LILO. The command for installing GRUB is `grub-install`. Also, you must specify the boot sector by device name when you install the boot loader. The basic command looks like

`grub-install /dev/hda`

   or

`grub-install '(hd0)'`

   Either command will install GRUB into the first sector (or the MBR) of your first hard drive. In the second example, you need single quotes around the device name. If you want to install GRUB in the boot sector of a partition rather than in the MBR, you would include a partition identifier, as in /dev/hda1 or (hd0,0).

   Remember that you do *not* need to reinstall GRUB after making changes to its configuration file! You'll only need to install GRUB in this way if you make certain changes to your disk configuration, such as resizing or moving the GRUB root partition, moving your entire installation to a new hard disk, or possibly reinstalling Windows (which tends to wipe out MBR-based boot loaders). In some of these cases, you may need to boot Linux via a backup boot loader, such as LILO or GRUB installed to floppy disk. (Type **`grub-install /dev/fd0`** to create one, and then label it and store it in a safe place.)

Interacting with GRUB

The first screen the GRUB boot loader shows you is a list of all of the operating systems that you specified with the `title` option in your GRUB configuration file. You can wait for the timeout to expire for the default operating system to boot. To select an alternative, you have to highlight the operating system that you want to have boot by using your arrow keys. Once your choice is highlighted, press the Enter key to start booting.

   Follow these steps when you want to change or pass additional options to your operating system:

1. Use your arrow keys to highlight the operating system that most closely matches what you want to boot.

2. Press the E key to edit this entry. You will now see a new screen listing all of the options for this entry.

3. Use your arrow keys to highlight the kernel option line.

4. Press the E key to edit the kernel options.

5. Edit the `kernel` line to add any options, such as 1 to boot to single user mode. GRUB will pass the extra option to the kernel.

6. Press the Enter key to complete the edits.

7. Press the B key to start booting.

   You can make whatever changes you like in step 5, such as using a different `init` program. You do this by appending `init=/bin/bash` (or whatever program you want to use) to the end of the `kernel` line.

Understanding the Boot Process

Any time that you modify the way your computer boots, there is the possibility that you will not get the results that you expect. In these cases, it is useful to know where you can turn to for more information on what is happening during startup. The reports you receive on a particular boot can better guide you once you understand something about what's *supposed* to happen when a Linux system boots.

Extracting Information on the Boot Process

Certain Linux kernel and module log information is stored in what is called the *kernel ring buffer*. By default, Linux displays messages destined for the kernel ring buffer during the boot process—they're those messages that scroll past too quickly to read. (Some distributions hide all but the earliest of these messages unless you select a special option during the boot process.) You may inspect this information with this command:

```
# dmesg
```

This command generates a lot of output, so you may want to pipe it through the `less` pager or redirect it to a file. Here are some examples of these commands:

```
# dmesg | less
# dmesg > boot.messages
```

> **NOTE** Many Linux distributions store the kernel ring buffer to `/var/log/dmesg` soon after the system boots. Because new information is logged to the kernel ring buffer as the system operates, and because the kernel ring buffer's size is finite, you may need to consult this log file to learn about the boot process once the system has been operating for a while.

Another source of logging information is from the system logger (`syslogd`). The most useful `syslogd` file to look at is usually `/var/log/messages`, but `/var/log/syslog` and other log files in `/var/log` can also hold useful information.

> **TIP** Some Linux distributions also log boot time information to other files. Debian uses a daemon called `bootlogd` that, by default, will log any messages that go to `/dev/console` to the `/var/log/boot` file. Fedora and Red Hat make use of `syslogd` services to log information to `/var/log/boot.log`.

The Boot Process

The process of taking an *x*86 computer from its initial state when the power is turned on to having a working operating system running is complex due to the way that modern personal computers have evolved. The steps that a computer goes through in order to boot an operating system are as follows:

1. The system is given power and a special hardware circuit causes the CPU to look at a predetermined address and execute the code that is stored in that location. The BIOS resides at this location, so the CPU runs the BIOS.

2. The BIOS code performs some tasks. These include checking for hardware, configuring hardware, and looking for a boot sector. This boot sector contains more code that needs to be loaded into memory in order to continue the boot process. This code is called a boot loader or, if it is a multi-staged loader, a primary boot loader.

> **NOTE** The boot loader may be found in the first sector of a hard drive (the MBR) or on the first sector of another device, such as a floppy disk or CD-ROM. In these cases, the location is simply referred to as a boot sector, not an MBR. The difference is that an MBR contains partition definitions in addition to the boot loader code.

3. When the boot loader takes over from the BIOS, if it is a multi-staged loader, it looks for a secondary loader. Some boot loaders go straight to a secondary loader in a hard drive's *active partition*. Other boot loaders, like LILO and GRUB, can do much more. For example, these boot loaders allow you to set extra boot-time options and load other boot loaders.

> **NOTE** Active partitions are simply partitions that are marked with a special flag. Some boot loaders and OSs, such as DOS and Windows, use the active flag as a way of determining which OS to boot. Linux and its boot loaders ignore the active flag for most purposes. That doesn't stop Linux from being able to mark a partition as active. Use disk partitioning tools such as `fdisk`, `cfdisk`, and `sfdisk` for this task. Each hard drive should have only one active partition.

4. The final goal of the boot loader is to find a kernel (Linux or otherwise), load it into memory, and execute it.

5. Once the Linux kernel takes over, it performs tasks such as initializing devices, mounting the root partition, and finally loading and executing the initial program for your system. By default, this is the program `/sbin/init`.

6. The initial program gets the process ID (PID) of 1 since it is the first program to run on the system. Assuming that `/sbin/init` is the initial program, it reads a file called `/etc/`

`inittab` to determine what other programs to run. Other programs usually include `getty` programs for console logins, system initialization scripts for mounting more partitions or starting up system services, and an X Display Manager (XDM) program to give you a graphical login.

How the `init` program and the initialization scripts work is covered next, in "Dealing with Runlevels and the Initialization Process."

> **NOTE** If you would like more details on this boot process, read `http://www` `.linuxdevcenter.com/pub/a/linux/excerpts/linux_kernel/how_computer_` `boots.html`. This page describes the process from the computer being powered up to the kernel being loaded and launching `/sbin/init`.

Dealing with Runlevels and the Initialization Process

Linux relies on *runlevels* to determine what features are available. Runlevels are numbered from 0 to 6, and each one is assigned a set of services that should be active. Upon booting, Linux enters a predetermined runlevel, which you can set. Knowing what these functions are, and how to manage runlevels, is important if you are to control the Linux boot process and ongoing operations. To this end, you must understand the purpose of runlevels, be able to identify the services that are active in a runlevel, be able to adjust those services, be able to check your default and current runlevels, and be able to change the default and current runlevels.

Runlevel Functions

Earlier in this chapter, I described single user mode. To get to this mode when booting Linux, you use the number 1, the letter S or s, or the word `single` as an option passed to the kernel by the boot loader. Single user mode is simply an available runlevel for your system. The available runlevels on most systems are the numbers 0 through 6. The letters S and s are synonymous with runlevel 1 as far as many utilities are concerned.

Runlevels 0, 1, and 6 are reserved for special purposes and the remaining runlevels are available for whatever purpose you or your Linux distribution provider decide. Table 6.1 summarizes the conventional uses of the runlevels. Other assignments—and even runlevels outside the range of 0 to 6—are possible, but such configurations are rare. (Gentoo, described shortly, uses an unusual runlevel system.) If you run into odd runlevel numbers, consult `/etc/inittab`—it defines them, and often contains comments explaining the various runlevels.

TABLE 6.1 Runlevels and Their Purposes

Runlevel	Purpose
0	This is a transitional runlevel, meaning that it's used to shift the system from one state to another. Specifically, it shuts down the system. On modern hardware, the system should completely power down. If not, you would be expected to either reboot the computer manually or power it off.
1, s, or S	Single user mode. What services, if any, are started at this runlevel varies by distribution. It's typically used for low-level system maintenance that could be impaired by normal system operation, such as resizing partitions.
2	On Debian systems, this is a full multi-user mode with X running and a graphical login. Most other distributions leave this runlevel undefined.
3	On Fedora, Mandrake, Red Hat, and most other distributions, this is a full multi-user mode with a console (non-graphical) login screen.
4	Usually undefined by default.
5	On Fedora, Mandrake, Red Hat, and most other distributions, this is the same behavior as runlevel 3 with the addition of having X run with an XDM (graphical) login.
6	Used to reboot the system. This runlevel is also a transitional runlevel. Your system will be completely shut down and then the computer will reboot automatically.

WARNING Don't configure your default runlevel to 0 or 6. If you do, your system will immediately shut down or reboot once it finishes powering up. Runlevel 1 could conceivably be used as a default, but chances are you want to use 2, 3, or 5 as your default runlevel, depending on your distribution and use for the system.

As a general rule, distributions have been drifting toward Red Hat's runlevel set; however, there are some exceptions and holdouts, such as Debian. Gentoo also deserves special attention. Although it uses numbered runlevels at its core, Gentoo builds upon this by enabling an arbitrary number of *named* runlevels. The default runlevel is called, appropriately enough, `default`. Gentoo's system permits you to create named runlevels for, say, connecting a laptop to half a dozen different networks, each with its own unique network configuration requirements. When you move from one network to another, simply enter the appropriate runlevel, as described later in "Changing Runlevels on a Running System."

Identifying the Services in a Runlevel

There are two main ways to affect what programs run when you enter a new runlevel. The first way is to add or delete entries in your /etc/inittab file. There are many entries in a typical /etc/inittab file, and except for a couple of special cases, inspecting or changing the contents of this file is best left to experts.

> Once all of the entries in /etc/inittab for your runlevel are executed, your boot process is complete and you can log in.

Basics of the */etc/inittab* File

Entries in /etc/inittab follow a simple format. Each line consists of four colon-delimited fields:

id:*runlevels*:*action*:*process*

Each of these fields has a specific meaning:

Identification code   The *id* field consists of a sequence of 1–4 characters that identifies its function.

Applicable runlevels   The *runlevels* field consists of a list of runlevels for which this entry applies. For instance, 345 means that the entry is applicable to runlevels 3, 4, and 5.

Action to be taken   Specific codes in the *action* field tell init how to treat the process. For instance, wait tells init to start the process once when entering a runlevel and to wait for the process's termination, and respawn tells init to restart the process whenever it terminates (which is great for login processes). Several other actions are available; consult the man page for inittab for details.

Process to run   The *process* field is the process to run for this entry, including any options and arguments that are required.

The part of /etc/inittab that tells init how to handle each runlevel looks like this:

```
# grep "rc [0-6]" /etc/inittab
l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
```

These lines begin with codes that begin with an l (that is, a lowercase letter *L*, not a number 1) followed by the runlevel number—for instance, l0 for runlevel 0, l1 for runlevel 1, and so on.

These lines specify scripts or programs that are to be run when the specified runlevel is entered. In the case of this example (which was taken from a Fedora 3 system), all of the scripts are the same (`/etc/init.d/rc`), but the script is passed the runlevel number as an argument. Some distributions, though, call specific programs for certain runlevels, such as `shutdown` for runlevel 0.

> The upcoming section "Checking and Changing Your Default Runlevel" describes how to tell `init` what runlevel to enter when the system boots.

The SysV Startup Scripts

The `/etc/init.d/rc` script performs the crucial task of running all of the scripts associated with the runlevel. These scripts are stored in `/etc/rc.d/rc?.d`, `/etc/init.d/rc?.d`, `/etc/rc?.d`, or a similar location. (The precise location varies between distributions.) In all these cases, `?` is the runlevel number. When entering a runlevel, `rc` passes the `start` parameter to all the scripts with names that begin with a capital `S` and the `stop` parameter to all the scripts with names that begin with a capital `K`. These *System V (SysV) startup scripts* start or stop services depending on the parameter they're passed, so the naming of the scripts controls whether they're started or stopped when a runlevel is entered. These scripts are also numbered, as in `S10network` and `K35smb`. The `rc` program runs the scripts in numeric order. This feature enables distribution designers to control the order in which scripts run by giving them appropriate numbers. This control is important because some services depend on others. For instance, network servers must normally be started after the network is brought up.

In reality, the files in the SysV runlevel directories are symbolic links to the main scripts, which are typically stored in `/etc/rc.d`, `/etc/init.d`, or `/etc/init.d/rc.d` (again, the exact location depends on the distribution). These original SysV startup scripts have names that lack the leading `S` or `K` and number, as in `smb` instead of `K35smb`.

> You can also start services by hand. Run them with the `start` option, as in **`/etc/init.d/smb start`** to start the `smb` (Samba) server. Other useful options are `stop`, `restart` and `status`. Most scripts support all of these options.

To determine which services are active in a runlevel, examine the appropriate SysV startup script directory in search of scripts with filenames that begin with an `S`. Alternatively, you can use a runlevel management tool, as described next.

Managing Runlevel Services

The SysV startup scripts in the runlevel directories are symbolic links back to the original script. This is done so that you don't need to copy the same script into each runlevel directory. Instead, you can modify the original script without having to track down its copies in all of the SysV runlevel directories. You can also modify which programs are active in a runlevel by editing the link filenames. Numerous utility programs are available to help you manage these links, such as

chkconfig, ntsysv, update-rc.d, and rc-update. I describe the first two of these tools because they're supported on most distributions. If your distribution doesn't support these tools, you should check distribution-centric documentation.

Managing Runlevel Services with *chkconfig*

To list the services and their applicable runlevels with chkconfig, use the --list option. The output looks something like this but is likely to be much longer:

```
# chkconfig --list
pcmcia          0:off   1:off   2:on    3:on    4:on    5:on    6:off
nfs-common      0:off   1:off   2:off   3:on    4:on    5:on    6:off
xprint          0:off   1:off   2:off   3:on    4:on    5:on    6:off
setserial       0:off   1:off   2:off   3:off   4:off   5:off   6:off
```

This output shows the status of the services in all seven runlevels. For instance, you can see that nfs-common is inactive in runlevels 0–2, active in runlevels 3–5, and inactive in runlevel 6.

> **NOTE** On Red Hat, Fedora, and some other distributions, chkconfig can manage servers that are handled by xinetd as well as SysV startup scripts. The xinetd-mediated servers appear at the end of the chkconfig listing.

If you are interested in a specific service, you can specify its name:

```
# chkconfig --list nfs-common
nfs-common      0:off   1:off   2:off   3:on    4:on    5:on    6:off
```

To modify the runlevels in which a service runs, use a command like this:

```
# chkconfig --level 23 nfs-common on
```

> **NOTE** The previous example is for Debian-based systems. On Red Hat and similar systems, you would probably want to target runlevels 3, 4, and 5 with something like --level 345 rather than --level 23.

You can set the script to be on (to activate it), off (to deactivate it), or reset (to set it to its default value).

If you've added a startup script to the main SysV startup script directory, you can have chkconfig register it and add appropriate start and stop links in the runlevel directories. When you do this, chkconfig inspects the script for special comments to indicate default runlevels. If these comments are in the file and you are happy with the suggested levels, you can add it to these runlevels with a command like this:

```
# chkconfig --add nfs-common
```

This command adds the nfs-common script to those managed by chkconfig. You would, of course, change nfs-common to your script's name. This approach may not work if the script lacks the necessary comment lines with runlevel sequence numbers for chkconfig's benefit.

Managing Runlevel Services with *ntsysv*

The ntsysv utility is an interactive text-mode tool. It was created by Red Hat, and is used mainly on Red Hat and related distributions, such as Fedora and Mandrake. If you run ntsysv without any arguments, you can configure your current runlevel. If you want to specify alternate runlevels to configure, run it with the --level option, as in **ntsysv --level 1** to configure runlevel 1 or **ntsysv --level 23** to configure runlevels 2 and 3. In either event, the result resembles Figure 6.2, which shows ntsysv running within an xterm window.

FIGURE 6.2   The ntsysv program enables you to control services from a menu of available services.

To adjust services, use your keyboard's arrow keys to select a service and press the spacebar to toggle the service on or off. An asterisk in the square brackets indicates that the service is on. No asterisk means that the service will not execute for the runlevels you've selected. When you're done, press the Tab key to highlight the OK button and then press the Enter key.

Checking Your Runlevel

Sometimes it's necessary to check your current runlevel. Typically, you'll do this prior to changing the runlevel or to check the status if something's not working correctly. Two different runlevel checks are possible: checking your default runlevel and checking your current runlevel.

Checking and Changing Your Default Runlevel

You can determine your default runlevel by inspecting the /etc/inittab file with the less command or opening it in an editor. Alternatively, you may use the grep command to look

for the line specifying the `initdefault` action. On a Fedora system, you would see something like this:

```
# grep :initdefault: /etc/inittab
id:5:initdefault:
```

On Debian, you would probably see this:

```
# grep :initdefault: /etc/inittab
id:2:initdefault:
```

You may notice that neither system defines a process to run. In the case of the `initdefault` action, the process field is ignored.

If you want to change the default runlevel for the next time you boot your system, edit the `initdefault` line in `/etc/inittab` and change the runlevel field to the value that you want.

Determine Your Current Runlevel

If your system is up and running, you can determine your runlevel information with the `runlevel` command:

```
# runlevel
N 2
```

The first character is the previous runlevel. When the character is `N`, this means that the system has not switched runlevels since booting. It is possible to switch to different runlevels on a running system with the `init` and `telinit` programs, as described next. The second character in the `runlevel` output is your current runlevel.

Changing Runlevels on a Running System

Sometimes you may want to change runlevels on a running system. You might do this to get more services, such as going from a console to a graphical login runlevel, or to shut down or reboot your computer. This can be accomplished with the `init` (or `telinit`), `shutdown`, `halt`, `reboot`, and `poweroff` commands.

Changing Runlevels with *init* or *telinit*

The `init` process is the first process run by the Linux kernel, but you can also use it to have the system reread the `/etc/inittab` file and implement changes it finds there or to change to a new runlevel. The simplest case is to have it change to the runlevel you specify. For instance, to change to runlevel 1 (the runlevel reserved for single user or maintenance mode), you would type this command:

```
# init 1
```

To reboot the system, you can use `init` to change to runlevel 6 (the runlevel reserved for reboots):

init 6

A variant of `init` is `telinit`. This program can take a runlevel number just like `init` to change to that runlevel, but it can also take the Q or q option to have the tool reread `/etc/inittab` and implement any changes it finds there. Thus, if you've made a change to the runlevel in `/etc/inittab`, you can immediately implement that change by typing **telinit q**.

> The `man` pages for these commands indicate slightly different syntaxes, but `telinit` is usually a symbolic link to `init`, and in practice `init` responds just like `telinit` to the Q and q options.

Changing Runlevels with *shutdown*

Although you can shut down or reboot the computer with `init`, doing so has some problems. One issue is that it's simply an unintuitive command for this action. Another is that changing runlevels with `init` causes an immediate change to the new runlevel. This may cause other users on your system some aggravation because they'll be given no warning about the shutdown. Thus, it's better to use the `shutdown` command in a multi-user environment when you want to reboot, shut down, or switch to single user mode. This command supports extra options that makes it friendlier in such environments.

The `shutdown` program will send a message to all users who are logged into your system and prevent other users from logging in during the process of changing runlevels. The `shutdown` command also lets you specify when to effect the runlevel change so that users have time to exit editors and safely stop other processes they may have running.

When the time to change runlevels is reached, `shutdown` will signal the `init` process for you. In the most simple form, `shutdown` is invoked with a time argument like:

shutdown now

This will change the system to runlevel 1, the single user or maintenance mode. The `now` parameter causes the change to occur immediately. Other possible time formats include *hh:mm*, for a time in 24-hour clock format (such as 6:00 for 6:00 AM or 13:30 for 1:30 PM), and +*m* for a time *m* minutes in the future.

You can add extra parameters to specify that you want to reboot or halt (that is, power off) the computer. Specifically, `-r` reboots the system and `-h` (for halt) powers it off. For instance, you might type **shutdown -r +10** to reboot the system in 10 minutes.

To give people some warning about the impending shutdown, add a message to the end of the command:

shutdown -h +15 system going down for maintenance

If you schedule a shutdown but then change your mind, you can use the `-c` option to cancel it:

shutdown -c "never mind"

Changing Runlevels with *halt, reboot*, and *poweroff* Commands

Three additional shortcut commands are `halt`, `reboot`, and `poweroff`. (In reality, `reboot` and `poweroff` are usually symbolic links to `halt`. This command behaves differently depending on the name with which it's called.) As you might expect, these commands halt the system (shut it down without powering it off), reboot it, or shut it down and (on hardware that supports this feature) turn off the power, respectively.

   Ordinarily, `halt` and its variants call `shutdown` to do the bulk of their work. You can override this behavior with the `-f` option, which causes the program to force a shutdown without going through the usual steps taken by `shutdown`. This option can be handy if the system is misbehaving so badly that `shutdown` doesn't work.

> **WARNING**   Don't use the `-f` option on a routine basis to shut down the system. The `shutdown` command (or `init`, if you use it) goes through steps to cleanly stop programs, unmount filesystems, and so on. Typing **halt -f** shuts down the system in a somewhat less clean way, which could have negative consequences in the form of unsaved files or other problems.

Managing the Shell Environment

Once the computer boots or reboots, you're normally presented with a login prompt, whereupon you can begin using the computer. (Of course, servers can work without anybody using the console.) The simplest case is a text-mode login, which causes Linux to launch a *shell* program. (Shell programs are also launched by `xterm` and similar GUI command-prompt windows.) Shell programs accept commands that you type and in response take actions, such as launching programs or changing their own environment for the benefit of subsequent commands. Linux supports several shell programs, so knowing about a few of them can be helpful. You should know something about built-in shell functions and environment variables, both of which are useful tools when using shells. Environment variables may also be used to help control programs launched from the shell. Configuring shells requires knowing about the names and formats of their configuration files; you can then use shell functions and environment variables to configure the shell.

Shell Options

As with many key software components, Linux provides a range of options for shells. A complete list would be quite long, but the more common choices include the following:

bash   The GNU Bourne Again Shell (`bash`) is based on the earlier Bourne shell but extends it in several ways. In Linux, `bash` is the most common default shell for user accounts, and it's the one emphasized in this book and on the LPI exam.

bsh   The Bourne shell upon which bash is based also goes by the name bsh. It's not often used in Linux, although the bsh command is usually a symbolic link to bash.

tcsh   This shell is based on the earlier C shell (csh). It's a fairly popular shell in some circles, but no major Linux distributions make it the default shell. Although it's similar to bash in many respects, some operational details differ. For instance, you don't assign environment variables in the same way in tcsh as in bash.

csh   The original C shell isn't much used on Linux, but if a user is familiar with csh, tcsh makes a good substitute.

ksh   The Korn Shell (ksh) was designed to take the best features of the Bourne shell and the C shell and extend them further. It's got a small but dedicated following among Linux users.

zsh   The Z shell (zsh) takes shell evolution further than the Korn Shell, incorporating features from earlier shells and adding still more.

In addition to these shells, dozens more obscure ones are available. In Linux, most users run bash because it's the default. Some other OSs use csh or tcsh as the default, so if your users have backgrounds on non-Linux Unix-like OSs, they may be more familiar with these other shells. You can change a user's default shell by editing the account, as described in Chapter 8, "Administering the System."

The file /bin/sh is a symbolic link to the system's default shell—normally /bin/bash for Linux. This practice enables you to point to a shell (say, at the start of a simple shell script) and be assured that a shell will be called, even if the system's available shells change. This feature is particularly important when developing shell scripts that might be run on other computers. (The upcoming section "Linux Scripting" describes shell script creation and revision.)

Built-in Shell Functions

Shells accept typed commands and respond in various ways to those commands. Broadly speaking, shells accept two types of commands: internal and external. Internal commands are those that are implemented by the shell itself. External commands are programs that the shell launches. Many common Linux commands, such as ls and mv, are external commands. This means that these commands work the same way in any shell. Built-in shell commands, though, differ from one shell to another, although most shells implement a similar basic set of commands. Some of the most important internal commands in bash are as follows:

alias   This command creates an alternate means of accessing another command. For instance, typing **alias lls='ls -l'** causes the shell to respond to your subsequently typing **lls** as if you'd typed **ls -l**.

cd   The cd command, used to change the current directory, is one of the commands that's built into most shells. To use it, type the name of the target directory, as in **cd papers** to change into the papers directory.

exit   This command exits from a shell. It's generally used in place of logout to close xterm windows.

export   This command makes an *environment variable* available to programs launched from the shell. An environment variable is a piece of data addressable by name that describes something about the system or the current working environment. Environment variables are described in more detail shortly, in "Using Environment Variables."

logout   This command exits from login shells, much like `exit`. It won't work with non-login shells such as `xterm` windows, though.

set   In its most basic form, `set` displays a wide variety of options relating to `bash` operation. These options are formatted much like environment variables, but they aren't the same things. You can pass various options to `set` to have it affect a wide range of shell operations.

unset   This command removes an option from those used by `bash`, as displayed or modified by `set`.

These are just a few of the built-in `bash` functions. To learn about more, type **man builtins**. This command displays the `man` page for all of the built-in functions.

Using Environment Variables

People exist in certain environments—office buildings, homes, streets, forests, airplanes, and so on. These environments provide us with, among other things, certain information. For instance, we can tell by reading a street sign that we're at the intersection of State and Main Streets or that there's an old mill a short distance away. Just as we humans live in an environment, so do the programs we run on computers.

The features that are salient to people, though, aren't the same as the ones that are important to computer programs. Computer programs must be concerned with issues such as the amount of free disk space or the availability of memory. Linux also provides programs with a set of supplementary information known as environment variables. Like street signs, environment variables convey information about the resources available to the program. Therefore, understanding how to set and use environment variables is important for both system administrators and users.

Programs query environment variables to learn about the state of the computer as a whole, or what resources are available. These variables contain information such as the location of the user's home directory, the computer's Internet hostname, and the name of the command shell that's in use. Individual programs may also use program-specific environment variables to tell them where their configuration files are located, how to display information, or how to use other program-specific options. As a general rule, though, environment variables provide information that's useful to multiple programs. Program-specific information is more often found in program configuration files.

Where to Set Environment Variables

If you're using the `bash` shell, you can set an environment variable from a command prompt for a specific login by typing the variable name followed by an equal sign (=) and the variable's value and then typing **export** and the variable name on the next line. For instance, you could type the following:

```
$ NNTPSERVER=news.abigisp.com
$ export NNTPSERVER
```

The first line sets the environment variable in your shell, and the second makes it available to programs you launch from the shell. You can shorten this syntax to a single line by typing **export** at the start of the first line:

```
$ export NNTPSERVER=news.abigisp.com
```

The former syntax is sometimes preferable when setting multiple environment variables because you can type each variable on a line and then use a single export command to make them all available. This can make shorter line lengths than you would get if you tried to export multiple variables along with their values on a single line. For instance, you could type the following:

```
$ NNTPSERVER=news.abigisp.com
$ YACLPATH=/usr/src/yacl
$ export NNTPSERVER,YACLPATH
```

> **NOTE** This syntax is the same as that used for setting environment variables in shell configuration files, as described in "Shell Configuration Files." When setting environment variables in a shell script and configuration files, you should ignore the command prompts ($) shown in these examples.

The preceding examples assigned values to environment variables. In other contexts, though, the environment variable is preceded by a dollar sign ($). You can use this notation to refer to an environment variable when setting another. For instance, in bash, the following command adds :/opt/bin to the existing PATH environment variable:

```
$ export PATH=$PATH:/opt/bin
```

The Meanings of Common Environment Variables

You may encounter many common environment variables on your system. You can find out how environment variables are configured by typing **env**. This command is used to run a program with a changed set of environment variables, but when it is typed alone, it returns all the environment variables that are currently set, in a format similar to that of bash environment variable assignments:

```
NNTPSERVER=news.abigisp.com
```

Of course, the variables you see and their values will be unique to your system and even your account—that's the whole point of environment variables. Table 6.2 summarizes variables you may see in this output.

T A B L E 6 . 2   Common Environment Variables and Their Meanings

Variable Name	Explanation
USER	This is your current username. It's a variable that's maintained by the system.
SHELL	This variable holds the path to the current command shell.
PWD	This is the present working directory. This environment variable is maintained by the system. Programs may use it to search for files when you don't provide a complete pathname.
HOSTNAME	This is the current TCP/IP hostname of the computer.
PATH	This is an unusually important environment variable. It sets the *path* for a session, which is a colon-delimited list of directories in which Linux searches for executable programs when you type a program name. For instance, if PATH is /bin:/usr/bin and you type **ls**, Linux looks for an executable program called ls in /bin and then in /usr/bin. If the command you type isn't on the path, Linux responds with a command not found error. The PATH variable is typically built up in several configuration files, such as /etc/profile and the .bashrc file in the user's home directory.
HOME	This variable points to your home directory. Some programs use it to help them look for configuration files or as a default location in which to store files.
LD_LIBRARY_PATH	A few programs use this environment variable to indicate directories in which library files may be found. It works much like PATH.
PS1	This is the default prompt in bash. It generally includes variables of its own, such as \u (for the username), \h (for the hostname), and \W (for the current working directory). This value is frequently set in /etc/profile, but it is often overridden by users.
NNTPSERVER	Some Usenet news reader programs use this environment variable to specify the name of the news server system. This value might be set in /etc/profile or in the user's configuration files.
TERM	This variable is the name of the current terminal type. To move a text-mode cursor and display text effects for programs like text-mode editors, Linux has to know what commands the terminal supports. The TERM environment variable specifies the terminal in use. This information is combined with data from additional files to provide terminal-specific code information. TERM is normally set automatically at login, but in some cases you may need to change it.

TABLE 6.2   Common Environment Variables and Their Meanings *(continued)*

Variable Name	Explanation
DISPLAY	This variable identifies the display used by X. It's usually :0.0, which means the first (numbered from 0) display on the current computer. When you use X in a networked environment, though, this value may be preceded by the name of the computer at which you're sitting, as in machine4.threeroomco.com:0.0. This value is set automatically when you log in, but you may change it if necessary. You can run multiple X sessions on one computer, in which case each one gets a different DISPLAY number—for instance, :0.0 for the first session and :1.0 for the second.
EDITOR	Some programs launch the program pointed to by this environment variable when they need to call a text editor for you to use. Thus, changing this variable to your favorite editor can help you work in Linux. It's best to set this variable to a text-mode editor, though; GUI editors might cause problems if they're called from a program that was launched from a text-mode login.

> **WARNING**
>
> The PATH variable often includes the current directory indicator (.) so that programs in the current directory can be run. This practice poses a security risk, though, because a miscreant could create a program with the name of some other program (such as ls) and trick another user into running it by simply leaving it in a directory the victim frequents. Even the root user may be victimized in this way. For this reason, it's best to omit the current directory from the PATH variable, especially for the superuser. If it's really needed for ordinary users, put it at the *end* of the path.

Any given system is likely to have several other environment variables set, but these are fairly esoteric or relate to specific programs. If a program's documentation says that it needs certain environment variables set, you can set them system-wide in /etc/profile or some other suitable file, or you can set them in user configuration files, as you deem appropriate.

Although you can see the entire environment by typing **env**, this output can be long enough to be intimidating. If you just want to know the value of one variable, you can use the echo command, which echoes what you type to the screen. If you pass it a variable name preceded by a dollar sign ($), echo returns the value of the variable. Here's an example:

```
$ echo $PS1
[\u@\h \W]\$
```

This command reveals that the PS1 environment variable is set to [\u@\h \W]\$, which in turn produces a bash prompt like [david@penguin homes]$.

EXERCISE 6.1

Changing Your *bash* Prompt

This exercise describes how to change your bash prompt to show the current time and number of jobs managed by the shell rather than whatever your distribution's default is. To accomplish this task, follow these steps:

1. Log into the Linux system as a normal user.

2. Launch an xterm from the desktop environment's menu system, if you used a GUI login method.

3. Type **export PS1="\T; \j jobs>** ". The backslash (\) is an escape character that denotes special data to be inserted into the prompt when used in the PS1 environment variable. A \T is expanded into the current time in 12-hour format, and \j is expanded into the number of jobs the shell manages. The man page for bash has a complete list of expansions the PS1 variable accepts. The result of typing this command should be an immediate change in your prompt to resemble something like 04:42; 0 jobs>.

4. Wait for a minute and then run a program in the background by typing its name and appending an ampersand (&). For instance, you might type **xeyes &** to run the xeyes program from an xterm. You should see the number of jobs increase, and the time should change.

5. To make this change permanent, edit the .bashrc file in your home directory. Load this file into your favorite editor and add a line to its end that reads export PS1="\T; \j jobs>". Save the file and exit from the editor.

6. To test your change to .bashrc, log out and then log back in again. Instead of your distribution's default prompt, you should see the new one.

7. If you don't like the new prompt, edit .bashrc again and delete the line you added in step 5.

Shell Configuration Files

Configuring shells requires editing shell configuration files. These files can be classified in a couple of ways. First, files may be global files that affect all users of a shell or local files that affect just one user. Second, files may be login files that are launched only by a login process (such as a text-mode console login) or non-login files that are launched by other processes (such as when starting an xterm window). The result is a 2×2 matrix of configuration files, as shown in Table 6.3. (This table shows only bash configuration files; consult your shell's documentation if you're using another shell.)

Precisely which of these files are used differs from one distribution to another. No matter the name, though, these files are shell scripts. Shell scripting is described in more detail later, in

TABLE 6.3   Common bash Configuration Files

Type of File	Login File Location	Non-Login File Location
Global	/etc/profile and files in /etc/profile.d	/etc/bashrc or /etc/bash.bashrc
User	~/.bash_login, ~/.profile, or ~/.bash_profile	~/.bashrc

"Linux Scripting," but most bash startup scripts contain a series of bash commands. These commands may include both built-in commands and external commands.

Just as shells have startup scripts, they may also have logout scripts—scripts that run when the user logs out. For bash, this script is ~/.bash_logout. Most distributions don't create this script as part of users' default home directories, but individual users can do so. The logout script might execute programs to clean up temporary directories, remove security keys from memory, or perform other tasks that are appropriate when a user logs out.

> **WARNING**   One problem with logout strips is that they may not work well when users log in multiple times. If you regularly have multiple sessions open, such as logins in multiple Linux virtual terminals, be careful about what you do in a logout script lest you wipe out important temporary files when you log out of one session.

Another bash configuration file is ~/.inputrc, which helps customize your keyboard configuration. It consists of lines that look like this:

```
M-Control-u: universal-argument
```

This line maps the Meta-Ctrl-U keystroke to the universal-argument action. The Meta key is usually the Esc key on *x*86 systems, and the universal-argument action is one of many possible actions defined by the readline library, which is one of the basic text-mode input libraries used by Linux.

In most cases, there's no need to adjust the ~/.inputrc file, because the default readline mappings work well for *x*86 systems with standard keyboards. If you find that certain keystrokes don't work the way they should in text mode, though, you may want to research this configuration file further.

> **NOTE**   X uses its own keyboard input routines, so ~/.inputrc doesn't affect programs run in X, even text-mode programs run inside xterm windows.

Linux Scripting

You'll do much of your work on a Linux system by typing commands at a shell prompt. As you use Linux, though, you're likely to find some of these tasks to be quite repetitive. If you need to add a hundred new users to the system, for instance, typing **useradd** a hundred times can be tedious. Fortunately, Linux includes a way to cut through the tedium: *shell scripts*. These are simple programs written in an interpreted computer language that's embedded in the Linux shell you use to type commands.

Most Linux systems use the bash shell by default, so shell scripts are often written in the bash shell scripting language, but the tcsh and other shell scripting languages are quite similar. In fact, it's not uncommon to see shell scripts that run in any common Linux shell. You're not restricted to running shell scripts written in your default shell, however; the first line of a shell script identifies the shell that should be used to run it.

> Many Linux startup scripts, including SysV startup scripts, are in fact shell scripts. Therefore, understanding shell scripting is necessary if you want to modify a Linux startup script.

> Like any programming task, shell scripting can be quite complex. Consequently, this chapter barely scratches the surface of what can be accomplished through shell scripting. Consult a book on the topic, such as *Learning the Bash Shell, 2nd Edition*, by Cameron Newham and Bill Rosenblatt (O'Reilly, 1998), for more information.

To use a shell script, you must first know how to start one. Once you start one, you'll find that one of the easiest tasks to do is to call external commands. More advanced tasks include using variables and using conditional expressions.

Beginning a Shell Script

Shell scripts are plain-text files, so you create them in text editors. A shell script begins with a line that identifies the shell that's used to run it, such as the following:

```
#!/bin/sh
```

The first two characters are a special code that tells the Linux kernel that this is a script and to use the rest of the line as a pathname to the program that's to interpret the script. Shell scripting languages use a hash mark (#) as a comment character, so the script utility itself ignores this line, although the kernel doesn't. On most systems, /bin/sh is a symbolic link that points to /bin/bash, but it could point to some other shell. Specifying the script as using /bin/sh guarantees that any Linux system will have a shell program to run the script, but if the script uses

any features specific to a particular shell, you should specify that shell instead—for instance, use `/bin/bash` or `/bin/tcsh` instead of `/bin/sh`.

When you're done writing the shell script, you should modify it so that it's executable. You do this with the `chmod` command, as described in Chapter 4, "Managing Files and Filesystems." Specifically, you use the `+x` option to add execute permissions, probably in conjunction with **a** to add these permissions for all users. For instance, to make a file called `my-script` executable, you'd issue the following command:

```
$ chmod a+x my-script
```

You'll then be able to execute the script by typing its name, possibly preceded by `./` to tell Linux to search in the current directory for the script. If you fail to make the script executable, you can still run the script by running the shell program followed by the script name (as in **bash my-script**), but it's generally better to make the script executable. If the script is one you run regularly, you may want to move it to a location on your path, such as `/usr/local/bin`. When you do that, you won't have to type the complete path or move to the script's directory to execute it; you can just type **my-script**.

Using Commands

One of the most basic features of shell scripts is the ability to run commands. You can use both shell internal commands, such as those described earlier, in "Built-in Shell Functions," and external commands. Most of the commands you type in a shell prompt are in fact external commands—they're programs located in `/bin`, `/usr/bin`, and other directories on your path. You can run such programs, as well as internal commands, by including their names in the script. You can also specify parameters to such programs in a script. For instance, suppose you want to start a script that launches two `xterm` windows and the KMail mail reader program. Listing 6.3 presents a shell script that accomplishes this goal.

Listing 6.3: A Simple Script That Launches Three Programs

```
#!/bin/bash
/usr/bin/xterm &
/usr/bin/xterm &
/usr/bin/kmail &
```

Aside from the first line that identifies it as a script, the script looks just like the commands you might type to accomplish the task manually except for one fact: The script lists the complete paths to each program. This is usually not strictly necessary, but listing the complete path ensures that the script will find the programs even if the PATH environment variable changes. Also, each program-launch line in Listing 6.3 ends in an ampersand (&). This character tells the shell to go on to the next line without waiting for the first to finish. If you omit the ampersands in Listing 6.3, the effect will be that the first `xterm` will open but the second won't open until the first is closed. Likewise, KMail won't start until the second `xterm` terminates.

Although launching several programs from one script can save time in startup scripts and some other situations, scripts are also frequently used to run a series of programs that manipulate data in some way. Such scripts typically do *not* include the ampersands at the ends of the commands because one command must run after another or may even rely on output from the first. A comprehensive list of such commands is impossible because you can run any program you can install in Linux as a command—even another script. A few commands that are commonly used in scripts include the following:

Normal file manipulation commands The file manipulation commands, such as `ls`, `mv`, `cp`, and `rm`, are often used in scripts. You can use these commands to help automate repetitive file maintenance tasks.

grep This command is described in Chapter 1, "Linux Command-Line Tools." It locates files that contain specific strings.

find Where `grep` searches for patterns within the contents of files, `find` does so based on filenames, ownership, and similar characteristics. This command is described in Chapter 4, "Managing Files and Filesystems."

cut This command extracts text from fields in a file. It's frequently used to extract variable information from a file whose contents are highly patterned. To use it, you pass it one or more options that control what it cuts followed by one or more filenames. For instance, users' home directories appear in the sixth colon-delimited field of the `/etc/passwd` file. You could therefore type **cut -f 6 -d** ":" **/etc/passwd** to extract this information.

sed This program is described in Chapter 1. It provides many of the capabilities of a conventional text editor but via commands that can be typed at a command prompt or entered in a script.

echo Sometimes a script must provide a message to the user; `echo` is the tool to accomplish this goal. You can pass various options to `echo` or just a string to be shown to the user. For instance, echo "Press the Enter key" causes a script to display the specified string.

mail The `mail` command can be used to send e-mail from within a script. Pass it the `-s` *subject* parameter to specify a subject line and give it an e-mail address as the last argument. If used at the command line, you would then type a message and terminate it with a Ctrl+D. If used from a script, you might omit the subject entirely, pass it an external file as the message using input redirection, or use a here document to pass text to the `mail` command as input. (Chapter 1 describes input redirection and here documents.) You might want to use this command to send mail to the superuser about the actions of a startup script or a script that runs on an automated basis.

> Many of these commands are extremely complex, and completely describing them is beyond the scope of this chapter. You can consult these commands' man pages for more information.

Even if you have a full grasp of how to use some key external commands, simply executing commands you might type at a command prompt is of limited utility. Many administrative tasks require you to modify what you type at a command, or even what commands you enter, depending on information from other commands. For this reason, scripting languages include additional features to help you make your scripts useful.

Using Variables

Variables can help you expand the utility of scripts. A variable is a placeholder in a script for a value that will be determined when the script runs. Variables' values can be passed as parameters to scripts, generated internally to the scripts, or extracted from the script's environment.

Variables that are passed to the script are frequently called parameters. They're represented by a dollar sign ($) followed by a number from 0 up—$0 stands for the name of the script, $1 is the first parameter to the script, $2 is the second parameter, and so on. To understand how this might be useful, consider the task of adding a user. As described in Chapter 8, creating an account for a new user typically involves running at least two commands—`useradd` and `passwd`. You might also need to run additional site-specific commands, such as commands that create unusual user-owned directories aside from the user's home directory.

As an example of how a script with a parameter variable can help in such situations, consider Listing 6.4. This script creates an account and changes the account's password (you'll be prompted to enter the password when you run the script). It creates a directory in the `/shared` directory tree corresponding to the account, and it sets a symbolic link to that directory from the new user's home directory. It also adjusts ownership and permissions in a way that may be useful, depending on your system's ownership and permissions policies.

Listing 6.4: A Script That Reduces Account-Creation Tedium

```
#!/bin/sh
useradd -m $1
passwd $1
mkdir -p /shared/$1
chown $1.users /shared/$1
chmod 775 /shared/$1
ln -s /shared/$1 /home/$1/shared
chown $1.users /home/$1/shared
```

Chapter 8 describes account maintenance in more detail.

If you use Listing 6.4, you need type only three things: the script name with the desired username and the password (twice). For instance, if the script is called `mkuser`, you might use it like this:

```
# mkuser ajones
Changing password for user ajones
New password:
Retype new password:
passwd: all authentication tokens updated successfully
```

Most of the scripts' programs operate silently unless they encounter problems, so the interaction (including typing the passwords, which don't echo to the screen) is a result of just the `passwd` command. In effect, Listing 6.4's script replaces seven lines of commands with one. Every one of those lines uses the username, so by using this script, you also reduce the chance of an error.

Another type of variable is assigned within scripts themselves—for instance, they can be set from the output of a command. These variables are also identified by leading dollar signs, but they're typically given names that at least begin with a letter, such as `$Addr` or `$Name`. (When values are assigned to variables, the dollar sign is omitted, as illustrated shortly.) You can then use these variables in conjunction with normal commands as if they were command parameters, but the value of the variable is passed to the command.

For instance, consider Listing 6.5, which implements simple firewall rules using the `ipchains` utility. This script uses two variables. The first is `$ip`, which is extracted from the output of `ifconfig` using `grep` and `cut` commands. (The trailing backslash on the second line of the script indicates that the following line is a continuation of the preceding line.) When assigning a value to a variable from the output of a command, that command should be enclosed in back-quote characters (` ), which appear on the same key as the tilde (~) on most keyboards. These are *not* ordinary single quotes, which appear on the same key as the regular quote character (") on most keyboards. The second variable, `$ipchains`, simply points to the `ipchains` program. It could as easily be omitted, with subsequent uses of `$ipchains` replaced by the full path to the program. Variables like this are sometimes used to make it easier to modify the script in the future. For instance, if you move the `ipchains` program, you need only modify one line of the script. They can also be used in conjunction with conditionals to ensure that the script works on more systems—for instance, if `ipchains` were called something else on some systems.

Listing 6.5: Script Demonstrating Assignment and Use of Variables

```
#!/bin/sh
ip=`ifconfig eth0 | grep inet | cut -f 2 -d ":" | \
    cut -f 1 -d " "`
ipchains="/sbin/ipchains"
echo "Restricting access to $ip"
$ipchains -A input -p tcp -s 0/0 -d $ip 25 -j REJECT
$ipchains -A input -p tcp -s 0/0 -d $ip 80 -j REJECT
```

> **NOTE** Listing 6.5 is a poor firewall. It blocks only two ports and omits many other fea-tures useful in a firewall. It is, however, an accessible demonstration of the use of variables in a script.

Scripts like Listing 6.5, which obtain information from running one or more commands, are useful in configuring features that rely on system-specific information or information that varies with time. You might use a similar approach to obtain the current hostname (using the `hostname` command), the current time (using `date`), the total time the computer's been run-ning (using `uptime`), free disk space (using `df`), and so on. When combined with conditional expressions (described shortly), variables become even more powerful because then your script can perform one action when one condition is met and another in some other case. For instance, a script that installs software could check free disk space and abort the installation if there's not enough disk space available.

One special type of variable was mentioned earlier in this chapter: environment variables, described in "Using Environment Variables." Environment variables are assigned and accessed just like shell script variables. The difference is that the script or command that sets an environ-ment variable uses the `export` command (in `bash`) to make the value of the variable accessible to programs launched from the shell or shell script that made the assignment. Environment vari-ables are most often set in shell startup scripts, but the scripts you use can access them. For instance, if your script calls X programs, it might check for the presence of a valid $DISPLAY environment variable and abort if it finds that this variable isn't set. By convention, environ-ment variable names are all uppercase, whereas nonenvironment shell script variables are all lowercase or mixed case.

Using Conditional Expressions

Scripting languages support several types of *conditional expressions*. These enable a script to perform one of several actions contingent on some condition—typically the value of a variable. One common command that uses conditional expressions is `if`, which allows the system to take one of two actions depending on whether some condition is true. The `if` keyword's conditional expression appears in brackets after the `if` keyword and can take many forms. For instance, `-f file` is true if *file* exists and is a regular file; `-s file` is true if *file* exists and has a size greater than 0; and *string1 = string2* is true if the two strings have the same values.

To better understand the use of conditionals, consider the following code fragment:

```
if [ -s /tmp/tempstuff ]
   then
      echo "/tmp/tempstuff found; aborting!"
      exit
fi
```

This fragment causes the script to exit if the file /tmp/tempstuff is present. The then keyword marks the beginning of a series of lines that execute only if the conditional is true, and fi (if backwards) marks the end of the if block. Such code might be useful if the script creates and then later deletes this file, since its presence indicates that a previous run of the script didn't succeed.

An alternative form for a conditional expression uses the test keyword rather than square brackets around the conditional:

```
if test -s /tmp/tempstuff
```

You can also test a command's return value by using the command as the condition:

```
if [ command ]
   then
      additional-commands
fi
```

In this example, the *additional-commands* will be run only if *command* completes successfully. If *command* returns an error code, the *additional-commands* won't be run.

Conditional expressions are sometimes used in *loops* as well. Loops are structures that tell the script to perform the same task repeatedly until some condition is met (or until some condition is no longer met). For instance, Listing 6.6 shows a loop that plays all the .wav audio files in a directory.

Listing 6.6: A Script That Executes a Command on Every Matching File in a Directory

```
#!/bin/bash
for d in `ls *.wav` ;
   do play $d ;
done
```

The for loop as used here executes once for every item in the list generated by ls *.wav. Each of those items (filenames) is assigned in turn to the $d variable and so is passed to the play command.

Another type of loop is the while loop, which executes for as long as its condition is true. The basic form of this loop type is like this:

```
while [ condition ]
do
   commands
done
```

The until loop is similar, in form, but it continues execution for as long as its condition is *false*—that is, until the condition becomes true.

Using Functions

A *function* is a part of a script that performs a specific sub-task and that can be called by name from other parts of the script. Functions are defined by placing parentheses after the function name and enclosing the lines that make up the function within curly braces:

```
myfn() {
   commands
}
```

The keyword `function` may optionally precede the function name. In either event, the function is called by name as if it were an ordinary internal or external command.

Functions are very useful in helping to create modular scripts. For instance, if your script needs to perform half a dozen distinct computations, you might place each computation in a function and then call them all in sequence. Listing 6.7 demonstrates the use of functions in a simple program that copies a file but aborts with an error message if the target file already exists. This script accepts a target and a destination filename and must pass those filenames to the functions.

Listing 6.7: A Script Demonstrating the Use of Functions

```
#/bin/bash

doit() {
   cp $1 $2
}

function check() {
  if [ -s $2 ]
     then
        echo "Target file exists! Exiting!"
        exit
  fi
}

check $1 $2
doit $1 $2
```

If you enter Listing 6.7 and call it `safercp`, you might use it like this, assuming the file `original.txt` exists and `dest.txt` does not:

```
$ ./safercp original.txt dest.txt
$ ./safercp original.txt dest.txt
Target file exists! Exiting!
```

The first run of the command succeeded because `dest.txt` did not exist. When the command was run a second time, though, the destination file did exist, so the program terminated with the error message.

Note that the functions are not run directly and in the order in which they appear in the script. They're run only when called in the main body of the script (which in Listing 6.7 consists of just two lines, each corresponding to one function call).

E X E R C I S E   6 . 2

Creating a Simple Script

Shell scripts are useful tools, and effectively creating them requires practice. This lab begins your exploration of shell scripts, but in the long run, you'll need to learn to design your own shell scripts. This lab presents a shell script that extracts the IP address of your network's router from the `route` command's output and then pings the router three times. To begin with this script, follow these steps:

1. Log into the Linux system as a normal user.

2. Launch an `xterm` from the desktop environment's menu system, if you used a GUI login method.

3. Start an editor and tell it to edit a file called `testscript`.

4. Type the following lines into the editor:

   ```
   #!/bin/bash
   ip=`route -n | grep UG | cut -b 17-32`
   ping -c 3 $ip
   ```

   Be sure that you've typed every character correctly; any mistake may cause the script to misbehave. One common error is mistyping the back-tick characters (`` ` ``) as ordinary single quote characters (`'`). Also, note that the range of characters in the `cut` command (17–32) is fixed and could conceivably change if the `route` command's output were to change with future versions of the command.

5. Save the file and exit from the editor.

6. Type **chmod a+x testscript** to add the executable bit to the file's permissions.

7. Type **./testscript** to run the script. If all goes well, you should see the system ping your network's router three times.

This exercise employs networking commands that are more fully described in Chapter 9. Consult it for details on the `route` and `ping` commands.

Summary

Linux kernel configuration and booting can be complex and intimidating to new administrators. Linux is a modular kernel, so you must know how to identify and manage kernel modules—how to identify loaded modules, load new ones for the hardware you use, and disable loaded modules. If modifying existing modules isn't adequate to get new hardware or a new kernel feature to work, you may need to recompile a kernel. This task requires adjusting a potentially very large number of options, so plan to spend an afternoon doing it. Once the kernel is recompiled, you must boot it, which involves adjusting your boot loader configuration. Details vary depending on the boot loader you're using—LILO or GRUB—but generally speaking, you can duplicate an existing entry and then modify it for your new kernel.

Once you reboot with your new kernel (or boot with an old one), managing the boot process can be important. The kernel normally runs the `init` program as the first process, and `init` in turn runs a series of SysV startup scripts to start critical services running. Tapping into this chain enables you to fine-tune your Linux system's default boot-time configuration, enabling or disabling components to best match your needs. To dig very deep into this process, though, you must understand shells and the shell scripts that are used to build the SysV startup scripts. This knowledge is also quite useful outside of the startup process; shell scripts can help automate many day-to-day administrative tasks, and of course a lot of your interactions with Linux are through the `bash` (or some other) shell.

Exam Essentials

Explain the function of Linux kernel modules.   Kernel modules are kernel drivers and other routines that reside in files that are separate from the main kernel file. Using kernel modules helps keep the size of the main kernel file manageable, which is most important for distribution maintainers.

Summarize the tools used to manage kernel modules.   The `lsmod` program displays the modules that are currently loaded. The `insmod` and `modprobe` programs are used to install modules, while `rmmod` removes modules that are loaded.

Describe why you might want to recompile your kernel.   Recompiling your kernel enables you to upgrade to the latest kernel and optimize your kernel for your system. Locally compiled kernels can include precisely those drivers and other features that you need and can also be compiled with CPU features to suit your system.

Summarize the kernel compilation process   To compile a kernel, you must download the kernel source code, extract the source code, type **`make xconfig`** (or a similar command) to configure the kernel for your system, type **`make`** to build the kernel, type **`make modules_ install`** to install the kernel modules, and then copy the kernel file to a convenient location (typically `/boot`). You must then reconfigure your boot loader to boot the new kernel and reboot the system.

Explain how LILO is configured and used.   LILO uses the /etc/lilo.conf configuration file. This file contains global options and per-image options. Use the lilo program to install the LILO boot loader in your boot sector. When the system boots, LILO presents a boot: prompt at which you type the name of the kernel or OS you want to boot.

Describe how GRUB is configured and used.   GRUB uses the menu.lst or grub.conf configuration file in /boot/grub. This file contains global and per-image options. Use the grub-install program to install the boot loader. When GRUB boots, it presents a menu of OS options that you select using keyboard arrow keys.

Describe the boot process.   The CPU runs the BIOS, the BIOS loads and runs a boot loader, the boot loader loads and runs secondary boot loaders (if needed) and the Linux kernel, the Linux kernel loads and runs the initial system program (init), and init starts the rest of the system services via SysV and other startup scripts. The BIOS looks for boot loaders in various boot sectors, including the MBR of a hard drive or the boot sector of a disk partition of floppy disk.

Summarize where to look for boot-time log information.   The dmesg command prints out logs from the kernel ring buffer, which holds boot-time and other kernel messages. Other useful log information can be found in /var/log/messages and other files in /var/log.

Summarize the role of /sbin/init.   The init program is responsible for starting many programs and services on your Linux operating system. This is done by running processes that are listed in /etc/inittab, including an rc script that runs the SysV initialization scripts.

Explain how runlevels are configured.   The default runlevel is specified with a line like id:2:initdefault: in the /etc/inittab file. Use commands such as chkconfig, update-rc.d, and ntsysv to change which services are started when switching to specific runlevels. Runlevels 0, 1, and 6 are reserved for shutdown, single user mode, and rebooting, respectively. Runlevels 3, 4, and 5 are the common user runlevels on Red Hat and most other distributions, and runlevel 2 is the usual user runlevel on Debian systems.

Describe how to change runlevels.   The programs init and telinit can be used to change to other runlevels. shutdown, halt, poweroff, and reboot are also useful when shutting down, rebooting, or switching to single user mode.

Explain the function of environment variables   Environment variables are used to store information on the system for the benefit of running programs. Examples include the PATH environment variable, which holds the locations of executable programs, and HOSTNAME, which holds the system's hostname.

Describe how a shell script can be useful.   A shell script combines several commands, possibly including conditional expressions, variables, and other programming features, to make the script respond dynamically to a system. Therefore, a shell script can reduce administrative effort by performing a series of repetitive tasks at one command.

Review Questions

1. What would you expect to see if you type **uname -a**?

    A. Information on the username of the current user, including the user's real name

    B. Information on the running system, such as the OS, the kernel version, and the CPU type

    C. Information on the computer's hostname, as it's defined locally

    D. Information on the loaded kernel modules, including the modules upon which they depend

2. What is the most important practical difference between `insmod` and `modprobe`?

    A. `insmod` unloads a single module, whereas `modprobe` loads a single module.

    B. `insmod` loads a single module, whereas `modprobe` loads a module and all those upon which it depends.

    C. `insmod` isn't a real Linux command, but `modprobe` loads a module and all those upon which it depends.

    D. `insmod` loads a single module, whereas `modprobe` displays information about modules.

3. You type the command **rmmod ide_core**, but the system responds with the message ERROR: Module ide_core is in use by via82cxxx,ide_cd,ide_disk. What is the meaning of this response?

    A. The via82cxxx, ide_cd, and ide_disk modules all rely on ide_core, so ide_core can't be unloaded without first unloading these other modules.

    B. The ide_core module relies on via82cxxx, ide_cd, and ide_disk modules, so they can't be unloaded without first unloading ide_core.

    C. The ide_core module is a core module, meaning that it can never be unloaded once it's loaded.

    D. The ide_core module is buggy or the rmmod utility is broken; it should never return an error message.

4. What file or files might you edit to change the options that are automatically passed to kernel modules? (Select all that apply.)

    A. Files in the /etc/modules.d directory

    B. modules.dep in the modules directory

    C. .config in the kernel directory

    D. /etc/modules.conf

5. Which of the following commands would you type to configure a Linux kernel using an interactive text-mode tool?

    A. **make xconfig**

    B. **make menuconfig**

    C. **make config**

    D. **make textconfig**

6. Which of the following kernel features should you compile into the main kernel file of a regular disk-based installation to simplify booting the system? (Select all that apply.)

   A. Drivers for your boot disk's ATA controller or SCSI host adapter

   B. Support for your root (/) filesystem

   C. Drivers for your RS-232 serial port

   D. Drivers for your CD-ROM drive

7. What is the conventional name for a locally compiled Linux kernel on an *x*86 system?

   A. `/boot/kernel-`*version*

   B. `/boot/vmlinuz`

   C. `/boot/bzImage-`*version*

   D. `/usr/src/linux-`*version*

8. Where might the BIOS find a boot loader?

   A. RAM

   B. `/dev/boot`

   C. MBR

   D. `/dev/kmem`

9. Which of the following is the LILO boot loader configuration file?

   A. `/dev/lilo`

   B. The MBR

   C. `/boot/lilo/lilo.conf`

   D. `/etc/lilo.conf`

10. Which command is used to install GRUB into the MBR of your first ATA hard drive?

   A. `grub (hd0,1)`

   B. `grub-install /dev/hda1`

   C. `lilo /dev/hda`

   D. `grub-install /dev/hda`

11. What line in `/etc/inittab` would indicate that your default runlevel is *5*?

   A. `ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now`

   B. `id:5:initdefault:`

   C. `si:5:sysinit:/etc/init.d/rcS`

   D. `l5:5:wait:/etc/init.d/rc 5`

12. Which runlevels are reserved by `init` for reboot, shutdown, and single user mode purposes? (Select all that apply.)

   A. 0

   B. 1

   C. 5

   D. 6

13. You type the following command:

```
$ runlevel
5 3
```

What can you tell about your runlevel status? (Select all that apply.)

   A. The current runlevel is 5.

   B. The current runlevel is 3

   C. The previous runlevel is 5.

   D. The previous runlevel is 3.

14. Where is the best location for the current directory indicator (`.`) to reside in `root`'s PATH environment variable?

   A. Before all other directories.

   B. After all other directories.

   C. Nowhere; it shouldn't be in `root`'s path.

   D. Wherever is convenient.

15. You want to create a shortcut for the command `cd ~/papers/trade`. Which of the following lines, if entered in a `bash` startup script, will accomplish this goal?

   A. `alias cdpt='cd ~/papers/trade'`

   B. `export cdpt='cd ~/papers/trade'`

   C. `cd ~/papers/trade`

   D. `shortcut cdpt "cd ~/papers/trade"`

16. In what environment variable is the current working directory stored?

   A. PATH

   B. CWD

   C. PWD

   D. PRESENT

17. Which of the following commands, if typed in a bash shell, will create an environment variable called MYVAR with the contents mystuff?

A. `export MYVAR='mystuff'`

B. `MYVAR='mystuff'`

C. `$MYVAR==mystuff`

D. `echo $MYVAR mystuff`

18. After using a text editor to create a shell script, what step should you take before trying to use the script?

A. Set one or more executable bits using chmod.

B. Copy the script to the /usr/bin/scripts directory.

C. Compile the script by typing **bash *scriptname***, where *scriptname* is the script's name.

D. Run a virus checker on the script to be sure it contains no viruses.

19. Describe the effect of the following short script, cp1, if it's called as cp1 big.c big.cc:

```
#!/bin/sh
cp $2 $1
```

A. It has the same effect as the cp command—copying the contents of big.c to big.cc.

B. It compiles the C program big.c and calls the result big.cc.

C. It copies the contents of big.cc to big.c, eliminating the old big.c.

D. It converts the C program big.c into a C++ program called big.cc.

20. What is the purpose of conditional expressions in shell scripts?

A. They prevent scripts from executing if license conditions aren't met.

B. They display information on the script's computer environment.

C. They enable the script to take different actions in response to variable data.

D. They enable scripts to learn in a manner reminiscent of Pavlovian conditioning.

Answers to Review Questions

1. B. The `uname` command displays system information, and the `-a` option to that command causes it to display all the information it's designed to summarize. Options A, C, and D are all incorrect, although other utilities will produce these results: `whoami`, `hostname`, and `lsmod` do the jobs described by options A, C, and D, respectively.

2. B. Option B correctly summarizes the main actions of both of these commands. Option A states that `modprobe` loads a single module. This is only partially correct; `modprobe` loads the module you specify *and* those upon which it depends. Option A also states that `insmod` unloads a module, when in fact it loads a module. Option C says that `insmod` isn't a real command, but it is. Option D says that `modprobe` displays information about modules, but it doesn't; that function is handled by `modinfo`.

3. A. The `rmmod` utility can only remove modules that are not being used by other modules or by other system software. The error message in this case indicates that the other specified modules rely on `ide_core`, so you can't unload `ide_core` until that dependency is broken by unloading these other modules. Option B specifies this dependency backwards. Although some modules are hard to remove once loaded, they aren't called "core modules," as option C specifies. The `rmmod` utility can and does return error messages when you ask it to do something it can't do, contrary to what option D says.

4. A, D. The `/etc/modules.conf` file controls automatic module loading and options passed to modules automatically. On most distributions, you edit this file directly; however, some require you to edit files in `/etc/modules.d` and then type **modules-update** to have the system update `modules.conf` itself. The `modules.dep` file referred to in option B specifies module dependencies, not module options passed to the modules. The kernel's `.config` file controls what kernel features are compiled; it doesn't specify options passed to the modules.

5. B. The `make menuconfig` kernel configuration command starts a text-based interactive menu configuration tool. This tool provides the same capabilities as the X-based tool that's started by `make xconfig`. Both of these tools are more flexible than the bare-bones text-mode `make config`. There is no `make textconfig` target.

6. A, B. The Linux kernel needs to be able to access your hard disk to continue past the most basic boot stage, so it needs drivers for your hard disk's ATA controller or SCSI host adapter as well as support for whatever filesystem you use on your root (/) partition in the kernel itself. Alternatively, these drivers can be placed on an initial RAM disk, but this configuration requires more work. Both the RS-232 serial port and the CD-ROM drive aren't needed during the boot process. (CD-ROM drivers are, of course, needed for CD-ROM-based distributions, but the question specified a hard-disk-based installation.)

7. C. Locally-compiled Linux kernels are usually called `/boot/bzImage` or `/boot/bzImage-version`, where *version* is the version number. There is no standard `/boot/kernel-version` file. The `/boot/vmlinuz` or `/boot/vmlinuz-version` file is the stock distribution's kernel, not a locally compiled kernel. The `/usr/src/linux-version` name is held by the Linux kernel source directory, not a compile kernel file.

8. C. The Master Boot Record (MBR) can contain a boot loader that is up to 512 bytes in size. If more space is required, the boot loader would have to load a secondary boot loader. Although the boot loader is loaded into RAM, it's not stored there permanently because RAM is volatile storage. Both `/dev/boot` and `/dev/kmem` are references to files on Linux filesystems; they're meaningful only after the BIOS has found a boot loader and run it and lots of other boot processes have occurred.

9. D. Option D is the correct LILO configuration file. Option A is a fictitious file; it doesn't exist. Although LILO's boot loader code may be written to the MBR, as implied by option B, this isn't the location of the LILO configuration file. Option C is reminiscent of the location of the GRUB configuration file; it's not the location of the LILO configuration file.

10. D. You use `grub-install` to install the GRUB boot loader code into an MBR or boot sector. When using `grub-install`, you specify the boot sector on the command line. The MBR is the first sector on a hard drive, so you give it the Linux device identifier for the entire hard disk, `/dev/hda`. Option A specifies using the `grub` utility, which is an interactive tool, and the device identifier shown in option A is a GRUB-style identifier for what would probably be the `/dev/hda3` partition in Linux. Option B is almost correct but installs GRUB to the `/dev/hda1` partition's boot sector rather than the hard disk's MBR. Option C is the command to install LILO to the MBR rather than to install GRUB.

11. B. It is the `initdefault` action that specifies the default runlevel.

12. A, B, D. Runlevel 0 is the reserved runlevel for halting the system. Runlevel 1 is reserved for single user mode. Runlevel 6 is reserved for rebooting. Runlevel 5 is a regular, user-configurable runlevel. (Many systems use it for a regular boot with a GUI login prompt.)

13. B, C. The first number in the `runlevel` output is the previous runlevel (the letter N is used to indicate that the system has not changed runlevels since booting). The second number is the current runlevel.

14. C. The current directory indicator is particularly dangerous in `root`'s PATH environment variable because it can be used by unscrupulous local users to trick `root` into running programs of the unscrupulous user's design.

15. A. The `alias` built-in command creates a duplicate name for a (potentially much longer) command. Option A shows the correct syntax for using this built-in command; it causes the new alias `cdpt` to work like the much longer `cd ~/papers/trade`. The export command in option B creates an environment variable called `cdpt` that holds the value `cd ~/papers/trade`. This will have no useful effect. Option C, if placed in a `bash` startup script, will cause the user's current directory to shift to `~/papers/trade` immediately after the user logs in. There is no standard `shortcut` command, so option D is meaningless.

16. C. The PWD environment variable holds the present working directory. The PATH environment variable holds a colon-delimited list of directories in which executable programs are stored so that they may be run without specifying their complete pathnames. There are no standard CWD or PRESENT environment variables.

17. A. Option A creates the desired environment variable. Option B creates a local variable—but not an environment variable—called MYVAR holding the value mystuff. After typing option B, you could also type **export MYVAR** to achieve the desired goal, but option B by itself is insufficient. Option C isn't a valid bash shell command. Option D displays the contents of the MYVAR variable and also echoes mystuff to the screen, but it doesn't change the contents of any environment variable.

18. A. Scripts, like binary programs, normally have at least one executable bit set, although they can be run in certain ways without this feature. There is no standard /usr/bin/scripts directory, and scripts can reside in any directory. Scripts are interpreted programs, which means they don't need to be compiled. Typing **bash scriptname** will run the script. Viruses are extremely rare in Linux, and because you just created the script, the only ways it could possibly contain a virus would be if your system was already infected or if you wrote it as a virus.

19. C. The cp command is the only one called in the script, and that command copies files. Because the script passes the arguments ($1 and $2) to cp in reverse order, their effect is reversed—where cp copies its first argument to the second name, the cp1 script copies the second argument to the first name. The cp command has nothing to do with compiling C or C++ programs, so neither does the script.

20. C. Conditional expressions return a "true" or "false" response, enabling the script to execute one set of instructions or another or to terminate or continue a loop.

Chapter

7

Documentation and Security

THE FOLLOWING LINUX PROFESSIONAL INSTITUTE OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **1.108.1 Use and manage local system documentation (weight: 4)**
- ✓ **1.108.2 Find Linux documentation on the Internet (weight: 3)**
- ✓ **1.108.5 Notify users on system-related issues (weight: 1)**
- ✓ **1.114.1 Perform security administration tasks (weight: 4)**
- ✓ **1.114.2 Setup host security (weight: 3)**
- ✓ **1.114.3 Setup user level security (weight: 1)**

This book, although it's a good introduction to Linux, cannot cover everything Linux-related. For this reason, you should know how to learn more about Linux. Fortunately, most Linux programs ship with documentation in a couple of forms. You can also find a lot of help on the Internet. Knowing how to use both local and Internet-based documentation can make all the difference in getting a program (or even Linux as a whole) up and running, so I describe where to find such documentation and how to make effective use of it. Another documentation-related issue, particularly on large multi-user systems, is communicating important system events to users, so you should understand Linux's tools for handling these tasks.

In today's network environment, in which network break-ins are common, security is an important topic. This chapter covers several security issues: restricting access to the computer by port number, managing the security of individual programs, managing passwords, and setting miscellaneous account security options. Understanding these basics will help you begin to secure your computer.

> **WARNING** There is no such thing as a 100 percent secure computer. You can take steps to improve security, but no one step or set of steps will absolutely guarantee that you'll have no problems. You must decide for yourself (or the organization for which you work must decide) just how much effort to put into securing your systems and live with the level of threat that remains. This chapter's security information can help you start securing your system, but if you need more than very basic security, you'll have to learn and do more than I can describe here.

Using Local System Documentation

Local Linux documentation comes in several forms. One of the most accessible of these is manual pages (or man pages for short, after the utility used to read them). Typically, man pages are fairly terse reference materials, though. Most packages also ship with more tutorial-style documentation that's stored in any of several formats in the /usr/doc or /usr/share/doc directory tree.

Using Linux Manual Pages

The first Linux documentation system was introduced in Chapter 1, "Linux Command-Line Tools." The man program gives access to basic documentation on most Linux commands, file formats, system calls, and so on. In addition to man itself, a few additional man-related programs are available. All of these programs rely on some configuration features that you should understand.

Using *man* Pages

Table 1.1 in Chapter 1 summarizes the various manual sections, each of which holds a certain class of information, such as commands or file formats. As described in Chapter 1, you can use `man` by typing the command name followed by the topic:

```
$ man passwd
```

This example calls up the `man` page for the `passwd` command, which is used to change a user's password, as described later in this chapter (in "Passwords") and in Chapter 8, "Administering the System." You can include the section in the call in case a keyword applies to multiple sections:

```
$ man 5 passwd
```

This example displays information on the `/etc/passwd` file, because `man` section 5 covers file formats. The default command (**man passwd**) brings up the section 1 (commands and programs) page because section 1 comes before section 5 in the `MANSECT` configuration option, described later, in "Configuring the Manual System."

The convention for `man` pages is a succinct style that employs several headings. Common headings include the following:

Name   A `man` page begins with a statement of the command, call, or file that's described, along with a few words of explanation. For instance, the `man` page for `man` (section 1) has a Name heading that reads `man – an interface to the on-line reference manuals`.

Synopsis   The synopsis provides a brief description of how a command is used. This synopsis uses a summary format similar to that used to present synopses in this book, showing optional parameters in square brackets (`[]`), for instance.

Description   The description is an English-language summary of what the command, file, or other element does. The description can vary from a very short summary to something many pages in length.

Options   This area summarizes the options outlined in the Synopsis area. Typically, each option appears in a list, with a one-paragraph explanation indented just below it.

Files   This heading introduces files that are associated with the `man` page's subject. These might be configuration files for a server or other program, related configuration files for a configuration file, or what have you.

See also   This heading introduces pointers to related information in the `man` system, typically with a section number appended. For instance, `less(1)` refers to the section 1 `man` page for `less`.

Bugs   Many `man` pages provide a Bugs heading, after which the author describes any known bugs or states that no known bugs exist.

History   Some `man` pages provide a summary of the program's history, citing project start dates and major milestones between then and the current version. This history isn't nearly as comprehensive as the changes file that ships with most programs' source code.

Author   Most `man` pages end with an Author section, which tells you how to contact the author of the program.

Specific manual pages may contain fewer, more, or different parts than these. For instance, the Synopsis part is typically omitted from man pages on configuration files. If a man page has a particularly verbose description, it's often split into several parts, each with its own title.

Although man pages can be an extremely helpful resource, you must understand their purpose and limitations. Unlike the help systems in some OSs, Linux man pages are not supposed to be either comprehensive or tutorial in nature; they're intended as quick references to help somebody who's already at least somewhat familiar with a command. They're most useful when you need to know the options to use with a command, the format of a configuration file, or similar summary information. If you need to learn a new program from scratch, other documentation is often a better choice. The quality of man pages is also variable; some are very good, but others are frustratingly terse, and even occasionally inaccurate. For the most part, they're written by the programmers who wrote the software in question, and programmers seldom place a high priority on user documentation.

> **NOTE** Although man is a text-mode command, GUI variants exist. The xman program, for instance, provides a point-and-click method of browsing through man pages. You can't type a subject on the command line to view it as you would with man, though—you must launch xman and then browse through the manual sections to a specific subject.

Additional Help Programs

One of the problems with man pages is that it can be hard to locate help on a topic unless you know the name of the command, system call, or file you want to use. Fortunately, methods of searching the manual database exist and can help lead you to an appropriate man page:

Summary search   The whatis command searches summary information contained in man pages for the keyword you specify. The command returns a one-line summary (the Name section of the man page, in fact) for every matching man page. You can then use this information to locate and read the man page you need. This command is most useful for locating all the man pages on a topic. For instance, typing **whatis man** returns lines confirming the existence of the man page entries for man, in sections 1, 7, and 1p. (The precise hits returned by whatis will vary from one system to another.)

Thorough search   The apropos command performs a more thorough search, of both the Name and Description sections of man pages. The result looks much like the results of a whatis search except that it's likely to contain many more results. In fact, doing an apropos search on a very common word, such as the, is likely to return so many hits as to make the search useless. A search on a less common word is likely to be more useful. For instance, typing **apropos samba** returns fewer than a dozen entries, including those for cupsaddsmb, smbpasswd, and lmhosts—all tools related to the Samba file- and printer-sharing tool. (The exact number of hits returned by apropos will vary from system to system, depending on the packages installed.)

Configuring the Manual System

On most Linux systems, the man system works correctly as delivered; however, like most Linux tools, it's configurable. The man configuration file is /etc/man.conf, and like most Linux configuration files, you can edit it in a text editor. The default options work well, and as a general rule, you shouldn't idly edit these entries.

One type of man option you might want to adjust is the man path, which is the list of directories in which man searches for man files. This feature is set with a series of MANPATH entries:

```
MANPATH  /usr/share/man
MANPATH  /usr/local/share/man
MANPATH  /usr/X11R6/man
MANPATH  /usr/local/man
MANPATH  /usr/man
```

Each entry consists of one directory, and man searches for files in the order in which they're presented. Sometimes you might want to add directories—say, because you've installed a program that places man pages in an odd location, such as /opt/oddprog/man. You can add a MANPATH line pointing to this directory. You can even use the asterisk (*) as a wildcard, just as you can on the command line. For instance, /opt/*/man causes man to search every subdirectory of /opt for a man subdirectory.

> **NOTE** Actually, the directories you specify on the MANPATH lines contain subdirectories, one for each man section. (In some cases, additional subdirectories for man pages in non-English languages also exist.)

If you want to set the man path on a one-time basis, you can do so with the --path option, as in **man --path /opt/oddprog/man oddprog**. This command brings up the man page for oddprog, stored in the /opt/oddprog/man directory tree. Alternatively, you can run a specific file through the man system by specifying its complete path, as in **man /opt/oddprog/man/ man1/oddprog.1.gz**.

Another man feature you might want to adjust in the configuration file is the MANSECT line:

```
MANSECT  1:1p:8:2:3:3p:4:5:6:7:9:0p:tcl:n:l:p:o
```

This line tells man in which order it should search the manual sections. The default usually works fine, but if you or your users find you're constantly pulling up the wrong manual page because you use certain sections more than others, you can adjust this order to correct the problem.

Most man pages come in a compressed form. The /etc/man.conf file includes lines to map various filename extensions to the compression programs that created them:

```
.gz   /bin/gunzip -c
.bz2  /bin/bzip2 -c -d
.Z    /bin/zcat
```

Given these entries, man tries to pass files with extensions of .gz through the /bin/gunzip program, passing it the -c option. Files with .bz2 and .Z extensions are also supported by the bzip2 and zcat commands, respectively. In most cases, there's no need to adjust these mappings; however, if you install a program that uses a peculiar compression tool for its man pages, you must either uncompress and then recompress the man pages yourself or add a mapping in /etc/man.conf.

🌐 Real World Scenario

The *info* Page System

Some developers and organizations, including the Free Software Foundation (FSF; http://www.fsf.org), favor the info program to the man program. Like man pages, info pages contain the official reference manuals for programs. The info system, though, supports a hierarchical structure with hyperlinks, similar to web pages. In theory, this enables info pages to be larger without overwhelming their users and helps users to quickly locate information. In practice, the info browser can be a bit intimidating until you're used to it.

To use info pages, you should type the command followed by the topic, much as you do to use man pages, as in **info ls** to learn about the ls command. You can then move between nodes (that is, individual pages) by moving the cursor over a link (denoted by an asterisk) and pressing the Enter key or by using various keystrokes. The N, P, U, L, and T keys, in particular, move to the next topic, the previous topic, up in the hierarchy, to the last page you read, or to the top of a topic tree, respectively. The Q key exits from the info reader.

In addition to reading info pages, the info reader can display man pages, although it won't convert man pages into documents that follow the hierarchical info system structure. Thus, you can use the info reader in place of the man program if you like.

Using Package Documentation

Most programs ship with documentation that's more extensive than the man pages alone. In most cases, this documentation installs in subdirectories of /usr/doc, /usr/share/doc, and /usr/X11R6/doc and similar locations with doc in their path names. Most packages install documentation files in subdirectories named after themselves in these directories. For instance, you might have a /usr/share/doc/samba-3.0.10 directory holding Samba documentation. In most cases, these directory names include program version numbers, so they'll change when you upgrade the package.

> ✓ **TIP** If you're looking for documentation but can't find it, try using your package manager's file-listing option to search for the string doc. For instance, using rpm, you might type **rpm -ql mystery | grep doc** to obtain a list of files in the mystery package and search that list for any files that contain the string doc.

In most cases, the `/usr/share/doc` documentation consists of the README file and other miscellaneous files that ship with the program's source code. Package maintainers simply shunt these files into the program's `/usr/share/doc` subdirectory and forget about them. For this reason, this documentation is sometimes a bit odd; it describes the build process and often refers to the location of the program files in the `/usr/local` directory tree, which usually isn't where you'll actually find them.

On the other hand, this additional documentation sometimes includes files in the Hypertext Markup Language (HTML), PostScript, and other formats. Such documentation sometimes includes tutorial information not found in `man` pages, example configuration files, and other useful data.

Overall, `/usr/share/doc` files are the "mystery surprise" of the Linux documentation world. Sometimes they're next to useless, but other times they're just what you need to learn how to use a program. The only way to know which is the case is to peruse the files.

The `/usr/share/doc` files are usually installed with the main package. Some packages, though, deliver a separate documentation package. This practice is most common when large amounts of supplemental documentation are available for a program; delivering documentation in a separate package enables you to choose whether or not to devote the space to the documentation. To learn which is the case, use your package manager to identify the package to which the documentation subdirectory belongs, as in **rpm -qf /usr/share/ doc/samba-3.0.10** for RPM-based systems. If the result is clearly a documentation package, you can use your package manager to delete it. If the result is the main package for the program, you can delete the documentation directory using ordinary file-management tools, but only at the cost of invalidating that part of your package database. This problem won't be debilitating, but it does slightly degrade the value of the package database. In most cases, you're better off leaving the documentation files in place; they do no harm other than consuming a small amount of disk space.

Finding Linux Documentation and Help on the Internet

The Internet is both the greatest boon and the greatest hindrance to learning about anything, including Linux. The Internet is wonderful because it's a treasure trove of information. It's terrible because much of the information is outdated or just plain wrong and because it can be so hard to find the information that's available.

One of the best starting points in doing Internet research on Linux is the Linux Documentation Project (LDP; `http://tldp.org`). You'll find lots of documentation here, as you might guess by the name. Other online resources can also be valuable, though, so you should be aware of them.

The Linux Documentation Project

The LDP is dedicated to providing more tutorial information than is commonly available with most Linux programs. You'll find several types of information at this site:

HOWTOs   Linux HOWTO documents are short and medium-length tutorial pieces intended to get you up to speed with a topic or technology. In the past, smaller HOWTOs were classified separately, as mini-HOWTOs; however, the distinction between the two types of document has diminished greatly in recent years. HOWTOs have varying focus—some describe particular programs, whereas others are more task oriented and cover a variety of tools in service to the task. As the name implies, they're generally designed to tell you how to accomplish some goal.

Guides   Guides are longer documents, often described as book-length. (In fact, some of them are available in printed form.) Guides are intended as thorough tutorial or reference works on large programs or general technologies, such as Linux networking as a whole.

FAQs   A *Frequently Asked Question (FAQ)* is, as the name implies, a question that comes up often—or more precisely, in the sense of the LDP category, a question and an answer to it. LDP FAQs are organized into categories, such as the Ftape FAQ or the WordPerfect on Linux FAQ. Each contains multiple questions and their answers, often grouped in subcategories. If you have a specific question about a program or technology, an appropriate FAQ can be a good place to look first for an answer.

LDP documents vary greatly in their thoroughness and quality. Some (particularly some of the Guides) are incomplete; you can click on a section heading and see an empty page or a comment that the text has yet to be written. Some LDP documents are very recent, but others are outdated, so be sure to check the date of any document before you begin reading—if you don't, you might end up doing something the hard way, or in a way that no longer works. Despite these flaws, the LDP can be an excellent resource for learning about specific programs or about Linux generally. The better LDP documents are excellent, and even those of marginal quality often present information that's not obvious from `man` pages, `info` pages, or official program documentation.

Additional Online Help Resources

In addition to the LDP, several additional online resources exist to provide documentation and help. Each of these sources has its own unique strengths and weaknesses:

Program web pages   Most Linux programs have associated web pages, and these web pages frequently include documentation. This resource is particularly helpful if you want to evaluate competing products or read up on one before installing it. Program information included in package files often points you to a program's web page. For instance, type `rpm -qpi packagename.rpm` to find information on *packagename* in an RPM system.

Most Linux distributions include the LDP documents in one or more special documentation packages. Check your distribution's package list to locate them. If you have fast always-up Internet access, though, you might want to use the online versions of LDP documents because you can be sure they're the latest available. Those that ship with a distribution can be weeks or months out of date by the time you read them.

Vendor web pages   Companies often provide Linux information on their web pages. These are usually identical to the program web pages just described in the case of commercial programs, but sometimes they're not. For instance, you might be able to locate Linux information relating to a video card or printer on the manufacturer's website. Also, don't forget your distribution maintainer's website. These are particularly useful when dealing with distribution-specific issues.

Web searches   Random websites often host useful information, but the trick is in locating them. You can use search engines such as Google (`http://www.google.com`) and Yahoo! (`http://www.yahoo.com`) to search the Web for keywords you specify. With luck, this will turn up a helpful website.

Web forums   A web forum is a website that enables people to post messages to it. Many distributions support web forums in which users discuss issues related to the distribution. These forums can be very useful resources and are often the first place you'll see new problems discussed, as well as fixes for those problems.

Usenet newsgroups   Usenet news is a forum that's similar in concept to a web forum, but Usenet was around long before web forums. Usenet posts are similar to e-mail messages in many ways except that they're public. Most ISPs and organizations such as universities maintain Usenet news servers, and you as an individual can access these using Linux programs like `tin` (`http://www.tin.org`) or Pan (`http://pan.rebelbase.com`). The Google Groups website (`http://groups.google.com`) maintains a database of newsgroup postings, so you can search for help in the form of previous queries about your problem. You should probably do this before posting a new cry for help; a quick search often turns up an answer much more quickly than does a new posting. Usenet is the most open of the major forum systems, which unfortunately means that it attracts more in the way of verbally abusive posts and off-topic discussions.

Mailing lists   Mailing lists are similar to web forums and Usenet newsgroups except that discussions are carried out via e-mail messages. (In fact, some forums are carried in multiple forms, so you can choose which you prefer.) Check vendor websites, distribution websites, and the websites for specific programs for information on mailing lists related to the product in question.

IRC   *Internet Relay Chat (IRC)* is a system for real-time interactions with other users. It's similar to the instant messaging systems that are also quite popular, but IRC involves public discussions. To use IRC, you must install an IRC client, such as Xchat (`http://www.xchat.org`). You then point the client to an IRC server, which links with other servers in an IRC network,

and join an IRC channel (which is a specific discussion forum, similar to a Usenet newsgroup). Check `http://www.irchelp.org/irchelp/networks/` for information on IRC networks.

By taking advantage of any or all of these resources, you can learn a great deal about Linux in a short period of time or solve almost any Linux problem. Not all resources are appropriate for solving all problems, though.

> Even if you don't make extensive use of any given information resource, you should at least take some time to familiarize yourself with the *type* of information to be found in each of these sources. Knowing what information a resource can provide can be very valuable if and when you need to locate that type of information.

Communicating with Users

Sometimes the information doesn't need to flow to you; it needs to flow from you. Specifically, you may need to notify users of important system events, upcoming changes, and so on. Several mechanisms exist to facilitate such communication:

Local login messages   The `/etc/issue` file holds a message that's displayed above the `login:` prompt at a text-mode console. Typically, this message identifies the computer. It can contain variables that are replaced with the computer's hostname, the kernel version number, the time, and so on, as described shortly.

Network login messages   The `/etc/issue.net` file is similar to `/etc/issue`, but it holds information that's displayed by the Telnet server just before it presents the `login:` prompt to the remote Telnet client. This file does *not* influence logins via the Secure Shell (SSH) server, though.

Message of the day   The message of the day (MOTD) is a message that's stored in `/etc/motd`. This file, if it exists, typically holds information you want to communicate to users about upcoming events or changes, such as scheduled downtime for system upgrades or changes to system use policies. Because users see its contents only when they log in, it's not very useful for notifying users of events that will occur in a matter of minutes. Most text-mode login methods, including console logins, Telnet, and SSH, display the MOTD. GUI login methods typically don't display it, though.

Fortunes   Some system administrators like to spice up the login experience with a pithy saying. You can do this with the help of the `fortune` program. Add a call to this program to the `/etc/bashrc` file to give users a new saying each day. You should be sure to call `fortune` only within an `if` clause, though, as shown in Listing 7.1; a direct call will interfere with some programs, such as `scp`. Note that the `fortune` program doesn't ship with all distributions.

Shutdown alerts   One particularly important type of user communication is the shutdown alert. You can pass a message to all text-mode users when you shut down the system via the `shutdown` command, as described in Chapter 6. Specifically, text after all other parameters is

treated as a shutdown alert message. For instance, **shutdown -h +10** "**System going down for maintenance**" displays the message "System going down for maintenance" to all users with increasing frequency until the system shuts down.

Listing 7.1: Code to Call *fortune* from */etc/bashrc*

```
if [ $TERM != "dumb" ]; then
    fortune
fi
```

The /etc/issue and /etc/issue.net files support variables that you can use to substitute information that might vary from one login to another or from one system to another (thus enabling you to use one file on multiple systems). These include \n (the computer's hostname), \r (the kernel version number), \s (the OS name—Linux), \m (the platform, such as x86), and \t (the time when the message is printed).

> **WARNING**   Be sure not to put too much information in system messages, and particularly in /etc/issue and /etc/issue.net. Advertising too much about your system, such as its kernel version number and platform, can give information to attackers that they can abuse to gain entry to your system.

One problem with all of these communication methods is that they're all geared toward text-mode users. If your system supports remote X-based users, these methods won't help you communicate with them. Likewise for users of servers, such as Web servers or mail servers. Of course, you can always employ more generic communication techniques, such as e-mail, instant messaging tools, bulletin boards, paper memos, and even word of mouth.

Restricting Access by Ports

Linux systems are often used as server computers, or at least they're connected to the Internet more or less directly. On such systems, network security is particularly important, because incorrectly configured servers can provide miscreants with a way into your system to do whatever damage they like. One of the first lines of defense against such problems is limiting access by *port*. In this context, a port is a numbered access point for your computer, much like a telephone extension number in a business phone system. (Chapter 9, "Basic Networking," describes networking concepts in more detail.) Ports are related to *sockets*, which are programming abstractions of network connection endpoints. Typically, you won't deal with sockets per se as a system administrator, though. You can protect access by port in three main ways: by configuring a firewall, by using restrictions built into super servers, and by disabling servers you're not actively using. First, though, you must know a bit about ports.

NOTE   The popular media uses the term *hacker* to refer to computer criminals. This word has an older meaning, though: It refers to individuals who are skilled with computers (and particularly with programming), who enjoy these activities, and who use their skills to productive and legal ends. Many Linux programmers consider themselves hackers in this positive sense. Therefore, I use another term, *cracker,* to refer to computer criminals.

Common Server Ports

Many firewalls and other network security devices operate by blocking or enabling access to specific ports. For instance, a firewall might block outside access to the SSH ports but let through traffic to the Simple Mail Transfer Protocol (SMTP) mail server port. In order to configure a firewall in this way, of course, you must know the port numbers. Linux systems contain a file, /etc/services, that lists service names and the ports with which they're associated. Lines in this file look something like this:

```
ssh             22/tcp      # SSH Remote Login Protocol
ssh             22/udp      # SSH Remote Login Protocol
telnet          23/tcp
# 24 - private
smtp            25/tcp
```

The first column contains a service name (ssh, telnet, or smtp in this example). The second column contains the port number and protocol (such as 22/tcp, meaning TCP port 22). Anything following a hash mark (#) is a comment and is ignored. The /etc/services file lists port numbers for both *Transmission Control Protocol (TCP)* and *User Datagram Protocol (UDP)* ports. (These two types of protocol are both important in TCP/IP networking; each carries different types of traffic.) Typically, a single service is assigned use of the same TCP and UDP port numbers (as in the ssh service in this example), although in practice most protocols use just one or the other. When configuring a firewall, it's generally best to block both TCP and UDP ports; this ensures you won't accidentally block the wrong port type.

Table 7.1 summarizes the port numbers used by the most important protocols run on Linux systems. This list is, however, incomplete; it only hits some of the most common protocols. In fact, even /etc/services is incomplete and may need to be expanded for certain obscure servers. (Their documentation describes how to do so, if necessary.)

NOTE   Table 7.1 shows the ports used by the *servers* for the specified protocols. In most cases, clients can and do use other port numbers to initiate connections. For instance, a mail client might use port 43411 on client.pangaea.edu to connect to port 143 on mail.pangaea.edu. Client port numbers are assigned by the kernel on an as-needed basis, so they aren't fixed. (Clients can request specific port numbers, but this practice is rare.)

TABLE 7.1   Port Numbers Used by Some Common Protocols

Port Number	TCP/UDP	Protocol	Example Server Programs
20 and 21	TCP	FTP	ProFTPd, WU FTPd
22	TCP	SSH	OpenSSH, `lsh`
23	TCP	Telnet	`in.telnetd`
25	TCP	SMTP	sendmail, Postfix, Exim, qmail
53	TCP and UDP	DNS	BIND
67	UDP	DHCP	DHCP
69	UDP	TFTP	`in.tftpd`
80	TCP	HTTP	Apache, `thttpd`
88	TCP	Kerberos	MIT Kerberos, Heimdal
109 and 110	TCP	POP (versions 2 and 3)	UW IMAP
111	TCP and UDP	Portmapper	NFS, NIS, other RPC-based services
113	TCP	auth/ident	`identd`
119	TCP	NNTP	INN, Leafnode
123	UDP	NTP	NTP
137	UDP	NetBIOS Name Service	Samba
138	UDP	NetBIOS Datagram	Samba
139	TCP	NetBIOS Session	Samba
143	TCP	IMAP 2	UW IMAP
161	UDP	SNMP	Net-SNMP
177	UDP	XDMCP	XDM, KDM, GDM
220	TCP	IMAP 3	UW IMAP

TABLE 7.1   Port Numbers Used by Some Common Protocols *(continued)*

Port Number	TCP/UDP	Protocol	Example Server Programs
389	TCP	LDAP	OpenLDAP
443	TCP	HTTPS	Apache
445	TCP	Microsoft DS	Samba
514	UDP	Syslog	`syslogd`
515	TCP	Spooler	BSD LPD, LPRng, `cups-lpd`
636	TCP	LDAPS	OpenLDAP
749	TCP	Kerberos Admin	MIT Kerberos, Heimdal
5800–5899	TCP	VNC via HTTP	RealVNC, TightVNC
5900–5900	TCP	VNC	RealVNC, TightVNC
6000–6099	TCP	X	X.org-X11, XFree86

One key distinction in TCP/IP ports is that between *privileged ports* and *unprivileged ports*. The former have numbers below 1024. Unix and Linux systems restrict access to privileged ports to root. The idea is that a client can connect to a privileged port and be confident that the server running on that port was configured by the system administrator and can therefore be trusted. Unfortunately, on today's Internet, this trust would be unjustified based solely on the port number, so this distinction isn't very useful. Port numbers above 1024 may be accessed by ordinary users.

Configuring a Firewall

The first line of defense in network security is a *firewall*. This is a computer that restricts access to other computers or software that runs on a single computer to protect it alone. Broadly speaking, two types of firewalls exist: *packet-filter firewalls,* which work by blocking or permitting access based on low-level information in individual data packets (such as source and destination IP addresses and ports), and *proxy filters*, which partially process a transaction (such as a web page access) and block or deny access based on high-level features in this transaction (such as the filename of an image in the web page). This chapter describes Linux's packet-filter firewall tools, which can be very effective at protecting a single computer or an entire network against certain types of attack.

Where a Firewall Fits in a Network

Traditionally, firewalls have been routers that block undesired network transfers between two networks. Typically, one network is a small network under one management, and the other network is much larger, such as the Internet. Figure 7.1 illustrates this arrangement. (More complex firewalls that use multiple computers are also possible.) Dedicated external firewalls are available and can be good investments in many cases. In fact, it's possible to turn an ordinary computer into such a device by using Linux—either with a special-purpose distribution like the Linux Embedded Appliance Firewall (`http://leaf.sourceforge.net`) or by using an ordinary distribution and configuring it as a router with firewall features.

FIGURE 7.1   Firewalls can selectively pass some packets but not others, using assorted criteria.
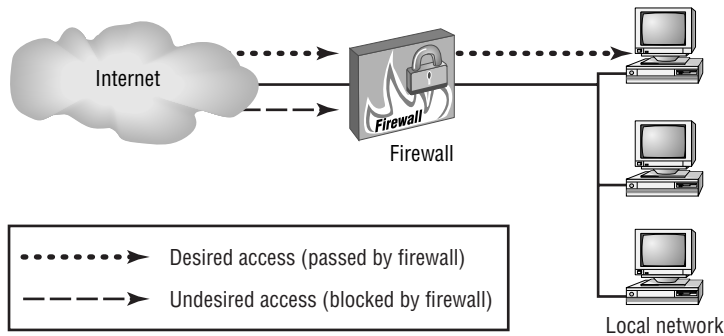
As described in more detail shortly, servers operate by associating themselves with particular network ports. Likewise, client programs bind to ports, but client port bindings aren't standardized. Packet filter firewalls block access by examining individual network packets and determining whether to let them pass based on the source and destination port number, the source and destination IP address, and possibly other low-level criteria, such as the network interface in a computer with more than one interface. For instance, in the arrangement in Figure 7.1, you might run a Samba file server internally, but outside computers have no business accessing that server. Therefore, you'd configure the firewall to block external packets directed at the ports used by Samba.

In addition to running a firewall on a router that serves an entire network, it's possible to run a firewall on an individual system. This approach can provide added protection to a sensitive computer, even if an external firewall protects that computer. It's also useful on computers that don't have the protection of a separate firewall, such as many broadband-connected systems.

Linux Firewall Software

Linux uses the `ipfwadm`, `ipchains`, and `iptables` tools to configure firewall functions. These tools are designed for the 2.0.*x,* 2.2.*x,* and 2.4.*x* kernels, respectively. The 2.6.*x* kernels con-

tinue to use the `iptables` tool as well. (The 2.4.*x* and later kernel series include the ability to use the older tools, but only as a compile-time option.) You can configure a firewall in any of several ways:

Manually   You can read up on the syntax of the tool used to configure your kernel and write your own script. This approach is described in the upcoming section, "Using `iptables`."

> For more information on this approach, consult a book on the subject, such as Robert L. Ziegler's *Linux Firewalls, 2nd Edition* (New Riders, 2001).

With the help of a GUI configuration tool   A few GUI configuration tools are available for Linux firewall configuration, such as Firestarter (`http://firestarter.sourceforge.net`) and Guarddog (`http://www.simonzone.com/software/guarddog`). Linux distributions often incorporate such tools as well, although the distribution-provided tools are often very simple. These tools let you specify certain basic information, such as the network port and the client and server protocols you wish to allow, and they generate firewall scripts that can run automatically when the system boots.

With the help of a website   Robert Ziegler, the author of *Linux Firewalls*, has made a website available that functions rather like the GUI configuration tools but via the Web. You enter information on your system, and the website generates a firewall script. This tool is available at `http://linux-firewall-tools.com/linux/`.

   If you use a GUI tool or website, be sure it supports the firewall tool your kernel requires. Most tools support `iptables`, and some support older tools or tools used in non-Linux OSs. Also, you shouldn't consider a firewall to be perfect protection. You might create a configuration that actually contains flaws, or flaws might exist in the Linux kernel code that implements the firewall rules.

> One of the advantages of a firewall, even to protect just one computer, is that it can block access attempts to *any* server. Most other measures are more limited. For instance, TCP Wrappers (described later, in "Using Super Server Restrictions") protects only servers configured to be run via TCP Wrappers from `inetd` and a few other servers that directly support TCP Wrappers. Passwords are good only to protect the servers that are coded to require them.

Using *iptables*

The `iptables` program is the utility that manages firewalls on recent Linux kernels (from 2.4.*x* through at least 2.6.*x*). Although these kernels can also use the older `ipchains` tool when so configured using kernel compile-time options, `iptables` is the more flexible tool and is therefore the preferred way of creating and managing packet-filter firewalls.

When using `iptables`, you should first understand how Linux's packet filter architecture works—you can create several types of rules, which have differing effects, so understanding how they interact is necessary before you begin creating rules. Actually creating the rules requires understanding the `iptables` command syntax and options. Finally, it's helpful to look at a sample firewall script and to know how it's installed and called by the system.

The Linux Packet Filter Architecture

In the 2.4.*x* and later kernels, Linux uses a series of "tables" to process all network packets it generates or receives. Each table consists of several "chains," which are series of pattern-matching rules—when a packet matches a rule, the rule can discard the packet, forward it to another chain, or accept the packet for local delivery. Figure 7.2 illustrates the `filter` table, which is the one you normally modify when creating a firewall. Other tables include the `nat` table, which implements Network Address Translation (NAT) rules, and the `mangle` table, which modifies packets in specialized ways.

FIGURE 7.2   Linux uses a series of rules, which are defined in chains that are called at various points during processing, to determine the fate of network packets.
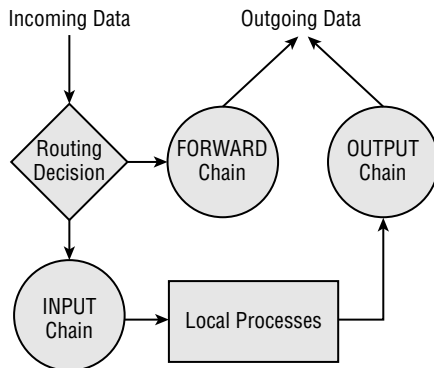
As shown in Figure 7.2, the `filter` table consists of three chains: INPUT, OUTPUT, and FORWARD. These chains process traffic destined to local programs, generated by local programs, and forwarded through a computer that's configured as a router, respectively. You can create rules independently for each chain. For instance, consider a rule that blocks all access directed at port 80 (the HTTP port, used by web servers) on any IP address. Applied to the INPUT chain, this rule blocks all access to a web server running on the local computer but doesn't affect outgoing traffic or traffic that's forwarded by a router. Applied to the OUTPUT chain, this rule blocks all outgoing traffic directed at web servers, effectively rendering web browsers useless, but it doesn't affect incoming traffic directed at a local web server or traffic forwarded by a router. Applied to the FORWARD chain, this rule blocks HTTP requests that might otherwise be forwarded by a computer that functions as a router but doesn't affect traffic from local web browsers or to local web servers.

Much of the task of creating a firewall involves deciding which chains to modify. Generally speaking, when you want to create a separate firewall computer (as illustrated in Figure 7.1), you modify the FORWARD chain (to protect the computers behind the firewall) and the INPUT chain (to protect the firewall system itself). When implementing a firewall to protect a server or workstation, you modify the INPUT chain and perhaps the OUTPUT chain. Blocking output packets can have the effect of preventing abuse of other systems or use of protocols you don't want being used. For instance, you might block outgoing traffic directed to a remote system's port 23, effectively disallowing use of Telnet clients on the system you're configuring.

All of the chains implement a default policy. This policy determines what happens to a packet if no rule explicitly matches it. The default for a default policy is ACCEPT, which causes packets to be accepted. This policy is sensible in low-security situations, but for a more secure configuration, you should change the default policy to DROP or REJECT. The former causes packets to be ignored. To the sender, it looks as if a network link was down. The REJECT policy causes the system to actively refuse the packet, which looks to the sender as if no server is running on the targeted port. This option requires explicit kernel support. Both DROP and REJECT have their advantages. DROP reduces network bandwidth use and reduces the system's visibility on the network, whereas REJECT can improve performance for some protocols, such as auth/ident, which may retry a connection in the event a packet is lost. Using either DROP or REJECT as a default policy means that you must explicitly open ports you want to use. This is more secure than using a default policy of ACCEPT and explicitly closing ports because you're less likely to accidentally leave a port open when it should be closed. Setting a default policy is described next, in "Creating Firewall Rules."

Creating Firewall Rules

To create firewall rules, you use the iptables command. You should probably start with the -L option, which lists the current configuration:

```
# iptables -L -t filter
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

You can obtain similar information from files in the /proc/net filesystem. For instance, ip_fwchains and ip_fwnames hold data on ipchains firewall rules; ip_tables_matches, ip_tables_names, and ip_tables_targets hold information relating to iptables firewalls; and ip_masquerade holds information on *IP masquerading*—a technique in which a router can "hide" a whole network from view, making all the systems look like one computer to the outside world. It's usually easier to use the iptables (or ipchains, for older systems) utility to access this information.

The -t filter part of this command specifies that you want to view the filter table. This is actually the default table, so you can omit this part of the command if you like. In any event, the result is a list of the rules that are defined for the specified (or default) table. In this case, no rules are defined and the default policy is set to ACCEPT for all three chains in the table. This is a typical starting point, although depending on your distribution and your installation options, it's possible yours will have rules already defined. If so, you should track down the script that sets these rules and change or disable it. Alternatively, or if you just want to experiment, you can begin by flushing the table of all rules by passing -F *CHAIN* to iptables, where *CHAIN* is the name of the chain. You can also use -P *CHAIN POLICY* to set the default policy:

```
# iptables -t filter -F FORWARD
# iptables -t filter -P FORWARD DROP
```

These two commands flush all rules from the FORWARD chain and change the default policy for that chain to DROP. Generally speaking, this is a good starting point when configuring a firewall, although using REJECT rather than DROP has its advantages, as described earlier. You can then add rules to the chain, each of which matches some selection criterion. To do so, you use an iptables command of the following form:

```
iptables [-t table] -A CHAIN selection-criteria -j TARGET
```

When modifying the filter table, you can omit the -t *table* option. The *TARGET* is the policy target, which can take the same values as the default policy (typically ACCEPT, REJECT, or DROP). In most cases, you'll use ACCEPT when the default policy is REJECT or DROP and REJECT or DROP when the default policy is ACCEPT. The *CHAIN* is, as you might expect, the chain to be modified (INPUT, OUTPUT, or FORWARD for the filter table). Finally, the *selection-criteria* can be one or more of several options that enable you to match packets by various rules:

Protocol The --protocol or -p option lets you specify the low-level protocol used. You pass the protocol name (tcp, udp, icmp, or all) to match packets of the specified protocol type. The all name matches all protocol types.

Source port The --source-port or --sport option matches packets that originate from the port number that you specify. (You can also provide a list of port numbers by separating them with colons, as in 1024:2048 to specify ports from 1024 to 2048, inclusive.) Note that the originating port number is the port number for the server program for packets that come from the server system, but it's the port number used by the client program for packets that come from the client system.

Destination port The --destination-port or --dport option works much like the --source-port option, but it applies to the destination of the packet.

Source IP address The --source or -s option filters on the source IP address. You can specify either a single IP address or an entire subnet by appending the netmask as a number of bits, as in -s 172.24.1.0/24.

Destination IP address The --destination or -d option works just like the --source option, but it filters based on a packet's destination address.

Input hardware interface   You can use the interface on which the packet arrives with the `--in-interface` or `-I` option, which accepts an interface name as an argument. For instance, `-I eth0` matches packets that arrive on the `eth0` interface. This option works with the INPUT and FORWARD chains but not with the OUTPUT chain.

Output hardware interface   The `--out-interface` or `-o` option works much like the `--in-interface` option, but it applies to the interface on which packets will leave the computer. As such, it works with the FORWARD and OUTPUT chains but not with the INPUT chain.

State   Network connections have states—they can be used to initiate a new connection, continue an existing connection, be related to an existing connection (such as an error message), or be potentially forged. The `--state` option can match based on these states, using codes of NEW, ESTABLISHED, RELATED, or INVALID. You must precede this option with the `-m state` option on the same `iptables` command line. This feature implements *stateful packet inspection,* which enables you to block connection attempts to certain ports while enabling you to initiate connections from those same ports. This feature is most useful in blocking connection attempts to unprivileged ports, thus denying crackers the ability to run unauthorized servers on those ports.

You can combine multiple items to filter based on several criteria. For instance, in a default-deny configuration, you can open traffic to TCP port 445 from the 172.24.1.0/24 network with a single command:

```
# iptables -A INPUT -p tcp --dport 445 -s 172.24.1.0/24 -j ACCEPT
```

In this case, the `selection-criteria` consist of three rules. Packets that match *all* of these rules will be accepted; those that fail to match even a single rule will be denied (assuming this is the default configuration) unless they match some other rule in the chain.

A complete chain is created by issuing multiple `iptables` commands, each of which defines a single rule. You can then view the result by typing **`iptables -L`**, as described earlier.

A Sample *iptables* Configuration

Because `iptables` creates a complete firewall configuration only through the use of multiple calls to the utility, Linux packet-filter firewalls are frequently created via shell scripts that repeatedly call `iptables`. (Chapter 6 describes shell scripts.) These scripts may be called as SysV startup scripts or in some other way as part of the startup procedure. For learning purposes, you may want to create a script that's not called in this way, though. Listing 7.2 shows a sample script that demonstrates the key points of firewall creation.

Listing 7.2: Sample Linux Firewall Script

```
#!/bin/bash

iptables -F INPUT
iptables -F FORWARD
iptables -F OUTPUT
```

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP

# Let traffic on the loopback interface pass
iptables -A OUTPUT -d 127.0.0.1 -o lo -j ACCEPT
iptables -A INPUT -s 127.0.0.1 -i lo -j ACCEPT

# Let DNS traffic pass
iptables -A OUTPUT -p udp --dport 53 -j ACCEPT
iptables -A INPUT -p udp --sport 53 -j ACCEPT

# Let clients' TCP traffic pass
iptables -A OUTPUT -p tcp --sport 1024:65535 -m state \
        --state NEW,ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p tcp --dport 1024:65535 -m state \
        --state ESTABLISHED,RELATED -j ACCEPT

# Let local connections to local SSH server pass
iptables -A OUTPUT -p tcp --sport 22 -d 172.24.1.0/24 -m state \
        --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -s 172.24.1.0/24 -m state \
        --state NEW,ESTABLISHED,RELATED -j ACCEPT
```

Listing 7.2 consists of three broad parts. The first three calls to `iptables` clear out all preexisting firewall rules. This is particularly important in a script that you're creating or debugging because you don't want to simply add new rules to existing ones; the result would likely be a confusing mishmash of old and new rules. The next three calls to `iptables` set the default policy to DROP on all three chains. This is a good basic starting point for a firewall. The remaining calls to `iptables` configure Linux to accept specific types of traffic:

Loopback traffic   The script sets the system to accept traffic to and from the loopback interface (that is, 127.0.0.1). Certain Linux tools expect to be able to use this interface, and because it's purely local, the security risk in accepting such traffic is very slim. Note that the lines that enable this access use both the IP address (via the -d and -s options) and the lo interface name (via the -o and -i options). This configuration protects against spoofing the loopback address—an attacker pretending to be 127.0.0.1 from another computer. This configuration, like most `iptables` configurations, requires two `iptables` rules: one to enable incoming traffic and one to enable outgoing traffic.

DNS traffic   The second block of rules enables UDP traffic to and from port 53, which handles DNS traffic. A configuration like this one is necessary on most systems to enable the computer to use its local DNS server. You could strengthen this configuration by

specifying only your local DNS server's IP address. (If you have multiple DNS servers, you'd need one pair of rules for each one.)

Client traffic   Listing 7.2 enables TCP packets to be sent from unprivileged ports (those used by client programs) to any system. This configuration uses stateful inspection to enable new, established, or related outgoing traffic but to allow only established or related incoming traffic. This configuration effectively blocks the ability to run servers on unprivileged ports. Thus, an intruder or malicious authorized user won't be able to log into an unauthorized server that runs on such a port—at least not without root access to change the configuration.

SSH server traffic   The final block of options enables access to the SSH server (TCP port 22). This access, though, is restricted to the 172.24.1.0/24 network (presumably the local network for the computer). This configuration uses stateful packet inspection to outgoing traffic from the SSH server for established and related data but not for new or invalid packets. Incoming packets to the server are permitted for new, existing, or related traffic, but not for invalid packets.

A configuration such as the one in Listing 7.2 is suitable for a workstation that runs an SSH server for remote administration but that otherwise runs no servers. For a computer that runs many servers, you might need to add several additional blocks of rules similar to the SSH block, each one customized to a particular server. For a dedicated router with firewall features, the emphasis would be on the FORWARD chain rather than the INPUT and OUTPUT chains, although such a system would likely need to perform some INPUT and OUTPUT chain configuration to support its own administration and use.

EXERCISE 7.1

Create a Firewall Script

This exercise guides you through the process of creating a simple iptables firewall script. The goal of this script is to protect a network of workstations; the firewall runs on a small router that handles this network's traffic. (This system should already be configured as a router.) Because the network being protected runs no servers that should be accessible from outside sites, all incoming connection attempts are blocked, but outgoing connection attempts are permitted. To configure this firewall, follow these steps:

1.   Log into the Linux system as a normal user.

2.   Launch an xterm from the desktop environment's menu system, if you used a GUI login method.

3.   Acquire root privileges. You can do this by typing **su** in an xterm, by selecting Session ➢ New Root Console from a Konsole, or by using sudo (if it's configured) to run the following commands.

4.   Create a firewall script file using your favorite editor. This file is called /usr/local/firewall.

5.   Type the code after step 10 into the firewall file as the firewall script. Change the value of the LocalNet variable for your network.

6. Save the firewall script and exit from the editor.

7. Make the firewall script executable by typing **chmod a+x /usr/local/firewall**.

8. Run the firewall script by typing **/usr/local/firewall**.

9. Verify that the firewall rules are active by typing **iptables -L**.

10. Configure the system to run the firewall script at startup. This can be done by calling the script from a local startup script (such as /etc/rc.d/rc.local), by creating a new SysV startup script that calls the firewall script, or by replacing an existing firewall startup script with a call to the new one.

```
#!/bin/bash


LocalNet=172.24.21.0/24


iptables -F INPUT

iptables -F FORWARD

iptables -F OUTPUT


iptables -P INPUT DROP

iptables -P FORWARD DROP

iptables -P OUTPUT DROP


# Let traffic on the loopback interface pass

iptables -A OUTPUT -d 127.0.0.1 -o lo -j ACCEPT

iptables -A INPUT -s 127.0.0.1 -i lo -j ACCEPT


# Let DNS traffic pass

iptables -A OUTPUT -p udp --dport 53 -j ACCEPT

iptables -A INPUT -p udp --sport 53 -j ACCEPT
```

```
# Let traffic from the local systems to the outside pass
iptables -A FORWARD -p tcp --sport 1025:65535 -s $LocalNet -j ACCEPT
iptables -A FORWARD -p udp --sport 1025:65535 -s $LocalNet -j ACCEPT
iptables -A FORWARD -p icmp -s $LocalNet -j ACCEPT


# Let only established incoming connections pass
iptables -A FORWARD -p tcp --dport 1025:65535 -d $LocalNet \
        -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -p udp --dport 1025:65535 -d $LocalNet \
        -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -p icmp -d $LocalNet \
        -m state --state ESTABLISHED,RELATED -j ACCEPT


# Let local connections to local SSH server pass
iptables -A OUTPUT -p tcp --sport 22 -d $LocalNet -m state \
        --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -s $LocalNet -m state \
        --state NEW,ESTABLISHED,RELATED -j ACCEPT
```

This firewall script configures forwarding (on the FORWARD chain) to pass TCP and UDP traffic from the local network only if it originates on unprivileged ports and to forward traffic to the local network only if it's destined to unprivileged ports and if it's not a new connection (the --state option omits the NEW option). ICMP traffic (used by ping and traceroute, among others) is also permitted to pass. This configuration also protects the computer on which it runs, but it provides a couple of exceptions. One opens traffic to and from DNS servers and the other opens traffic to and from the system's own SSH server, provided that traffic originates on the local network. These openings are required for administration of the router from anywhere but its own console.

Using *ipchains*

The 2.2.*x* kernel doesn't support `iptables`; it used `ipchains` as its firewall tool. Even if you're using a 2.4.*x* or later kernel, you might want to use `ipchains` because you have legacy firewall scripts that work to your satisfaction. When designing a new firewall, though, you should probably use `iptables` because it's more flexible.

If you must use `ipchains`, you should be aware of its major differences from `iptables`. For one thing, as the name implies, `ipchains` works on chains of firewall rules; these chains are not grouped into tables. In practice, `ipchains` works much like `iptables` on its default `filter` table; you can modify `input`, `output`, and `forward` chains.

Another important difference is that `ipchains` doesn't support stateful packet inspection (as implemented by the `--state` option to `iptables`). This feature is an important one for many firewalls, and is one of the reasons `iptables` is preferred to `ipchains`.

Many `ipchains` options are similar or identical to those of `iptables`. For instance, the `-P`, `-A`, `-p`, `-s`, `-d`, and `-j` options all work in more or less the same way in both tools. One notable difference is in the port specification (`--sport` and `--dport` in `iptables`). This information is folded into the `-s` and `-d` options in `ipchains`, as in `-d 192.168.27.72 23` to specify port 23 on 192.168.27.72.

Using Super Server Restrictions

Beyond firewalls, the first layer of access controls for many servers lies in the super server that launches the server in question. Chapter 9 describes the basics of configuring the `inetd` and `xinetd` super servers. You can use a package called TCP Wrappers with either super server, but it's more commonly used with `inetd`. The `xinetd` super server includes functionality that's similar to TCP Wrappers in its basic feature set.

> Whenever possible, apply redundant access controls. For instance, you can use both a firewall and TCP Wrappers or `xinetd` to block unwanted access to particular servers. Doing this helps protect against bugs and misconfiguration—if a problem emerges in the firewall configuration, for instance, the secondary block will probably halt the intruder. If you configure the system carefully, such an access will also leave a log file message that you'll see, so you'll be alerted to the fact that the firewall didn't do its job.

Controlling Access via TCP Wrappers

One popular means of running servers is via `inetd`, a server that listens for network connections on behalf of other servers and then launches the target servers as required. This approach can reduce the RAM requirements on a server computer when the server programs are seldom in use because only `inetd` need be running at all times. Chapter 9 covers `inetd` in more detail.

> **NOTE** Not all Linux systems use inetd. Fedora, Mandriva, Red Hat, and SuSE have all switched to xinetd, which includes its own access control features. TCP Wrappers isn't normally used in conjunction with xinetd.

One further advantage of inetd is that it can be used in conjunction with another package, known as TCP Wrappers. This package provides a program known as tcpd. Instead of having inetd call a server directly, inetd calls tcpd, which does two things: It checks whether a client is authorized to access the server, and if the client has this authorization, tcpd calls the server program.

TCP Wrappers is configured through two files: /etc/hosts.allow and /etc/hosts.deny. The first of these specifies computers that are allowed access to the system in a particular way, the implication being that systems not listed are not permitted access. By contrast, hosts.deny lists computers that are not allowed access; all others are granted access to the system. If a computer is listed in both files, hosts.allow takes precedence.

Both files use the same basic format. The files consist of lines of the following form:

```
daemon-list : client-list
```

The *daemon-list* is a list of servers, using the names for the servers that appear in /etc/services. Wildcards are also available, such as ALL for all servers.

The *client-list* is a list of computers to be granted or denied access to the specified daemons. You can specify computers by name or by IP address, and you can specify a network by using a leading or trailing dot (.) when identifying networks by name or IP address block, respectively. For instance, .luna.edu blocks all computers in the luna.edu domain, and 192.168.7. blocks all computers in the 192.168.7.0/24 network. You can also use wildcards in the *client-list*, such as ALL (all computers). EXCEPT causes an exception. For instance, when placed in hosts.deny, 192.168.7. EXCEPT 192.168.7.105 blocks all computers in the 192.168.7.0/24 network except for 192.168.7.105.

The man pages for hosts.allow and hosts.deny (they're actually the same document) provide additional information on more advanced features. You should consult them as you build TCP Wrappers rules.

> **WARNING** Remember that not all servers are protected by TCP Wrappers. Normally, only those servers that inetd runs via tcpd are so protected. Such servers typically include, but are not limited to, Telnet, FTP, TFTP, rlogin, finger, POP, and IMAP servers. A few servers can independently parse the TCP Wrappers configuration files, though; consult the server's documentation if in doubt.

Controlling Access via *xinetd*

In 2000 and 2001, the shift began from inetd to xinetd. Although xinetd *can* use TCP Wrappers, it normally doesn't because it incorporates similar functionality of its own. The distributions that use xinetd use a main configuration file called /etc/xinetd.conf, but this file is largely

empty because it calls separate files in the `/etc/xinetd.d` directory to do the real work. This directory contains separate files for handling individual servers. Chapter 9 includes information on basic `xinetd` configuration. For now, know that security is handled on a server-by-server basis through the use of configuration parameters, some of which are similar to the function of `hosts.allow` and `hosts.deny`:

Network interface   The `bind` option tells `xinetd` to listen on only one network interface for the service. For instance, you might specify `bind = 192.168.23.7` on a router to have it listen only on the Ethernet card associated with that address. This feature is extremely useful in routers, but it is not as useful in computers with just one network interface. (You can use this option to bind a server only to the loopback interface, 127.0.0.1, if a server should be available only locally. You might do this with a configuration tool like the Samba Web Administration Tool, or SWAT.) A synonym for this option is `interface`.

Allowed IP or network addresses   You can use the `only_from` option to specify IP addresses, networks (as in `192.168.78.0/24`), or computer names on this line, separated by spaces. The result is that `xinetd` will accept connections only from these addresses, similar to TCP Wrappers' `hosts.allow` entries.

Disallowed IP or network addresses   The `no_access` option is the opposite of `only_from`; you list computers or networks here that you want to blacklist. This is similar to the `hosts.deny` file of TCP Wrappers.

Access times   The `access_times` option sets times during which users may access the server. The time range is specified in the form *hour:min-hour:min*, using a 24-hour clock. Note that this option only affects the times during which the server will *respond*. If the `xinetd access_times` option is set to `8:00-17:00` and somebody logs in at 4:59 PM (one minute before the end time), that user may continue using the system well beyond the 5:00 PM cutoff time.

You should enter these options into the files in `/etc/xinetd.d` that correspond to the servers you want to protect. Place the lines between the opening brace ({) and closing brace (}) for the service. If you want to restrict *all* your `xinetd`-controlled servers, you can place the entries in the `defaults` section in `/etc/xinetd.conf`.

> **NOTE**   Some servers provide access control mechanisms similar to those of TCP Wrappers or `xinetd` by themselves. For instance, Samba provides `hosts allow` and `hosts deny` options that work much like the TCP Wrappers file entries, and NIS includes similar configuration options. These options are most common on servers that are awkward or impossible to run via `inetd` or `xinetd`.

Disabling Unused Servers

Quite a few server programs ship with most Linux distributions, which can be a great advantage—you don't need to go hunting for any servers you want to run. On the other hand, this very advantage can be a drawback—if you're not careful, you can end up running a server and

not even realize it's installed! For this reason, you should periodically search for servers and shut down any that you find that aren't really necessary. Several tools and techniques can help you find unused servers. These include perusing your configuration files, using local network activity tools, and using remote network scanners. Disabling unused servers can be done by uninstalling the package or by reconfiguring the server.

Examining Configuration Files

One way of spotting unused server programs is to examine your configuration files. On most systems, two classes of files are important: those controlling SysV startup scripts and those controlling your super server. Slackware is a bit odd because it doesn't use SysV startup scripts *per se;* instead, it uses one startup script for each runlevel. For Slackware, therefore, you must skim the relevant runlevel startup script looking for suspicious calls.

SysV startup scripts are described in detail in Chapter 6, so review that chapter for details of how they're managed. Generally speaking, you'll look in `/etc/rc?.d`, `/etc/init.d/rc?.d`, or `/etc/rc.d/rc?.d`, where *?* is your default runlevel number, for scripts whose names take the form S##*server*, where ## is a number and *server* is the name of the server. If you find such a script for a server you know you don't want to run, you should disable it using your SysV startup script editing tools, as described in Chapter 6.

Be aware that many SysV startup scripts start entire subsystems that aren't directly network-related. Thus, you'll probably see SysV startup scripts that you don't recognize. You shouldn't automatically disable these scripts because they may be necessary even if you don't recognize the name. If in doubt, leave it in place until you can research the matter further.

> Try doing a Web search on the name of the SysV startup script (minus the S and sequence number), possibly in conjunction with "Linux" or "startup script." Chances are you'll find a helpful reference.

The other major configuration file class you should examine is the super server configuration. These are described in detail in Chapter 9, so consult it for details concerning the `inetd` and `xinetd` configuration file formats. In both cases, you should look for servers you know shouldn't be running. Also, unlike SysV startup scripts, super servers only launch network servers. Therefore, you should take a more aggressive approach to disabling entries you don't recognize from your super server configuration than you do with SysV startup scripts.

Using Local Network Activity Tools

Beyond checking basic SysV and super server configuration, you can employ Linux tools to help find stray servers. One such tool is `netstat`. This program is the Swiss Army knife of network status tools; it provides many different options and output formats to deliver information on routing tables, interface statistics, and so on. For spotting unnecessary servers, you can use `netstat` with its `-a` and `-p` options, as shown here:

```
# netstat -ap
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
```

```
➥PID/Program name
tcp        0      0 *:ftp                    *:*                       LISTEN
➥690/inetd
tcp        0      0 teela.rodsbooks.com:ssh nessus.rodsbooks.:39361 ESTABLISHED
➥787/sshd
```

> **NOTE** I've trimmed most of the entries from this output to make it manageable as an example.

The `Local Address` and `Foreign Address` columns specify the local and remote addresses, including both the hostname or IP address and the port number or associated name from `/etc/services`. The first of the two entries shown here isn't actively connected, so the local address, the foreign address, and the port number are all listed as asterisks (*). This entry does specify the local port, though: `ftp`. This line indicates that a server is running on the `ftp` port (TCP port 21). The `State` column specifies that the server is listening for a connection. The final column in this output, under the `PID/Program name` heading, indicates that the process with a process ID (PID) of 690 is using this port. In this case, it's `inetd`.

The second output line indicates that a connection has been established between `teela.rodsbooks.com` and `nessus.rodsbooks.com` (the second hostname is truncated). The local system (`teela`) is using the `ssh` port (TCP port 22), and the client (`nessus`) is using port 39361 on the client system. The process that's handling this connection on the local system is `sshd`, running as PID 787.

It may take some time to peruse the output of `netstat`, but doing so will leave you with a much improved understanding of your system's network connections. If you spot servers listening for connections that you didn't realize were active, you should investigate the matter further. Some servers may be innocent or even necessary. Others may be pointless security risks.

> **TIP** When you use the -p option to obtain the name and PID of the process using a port, the `netstat` output is wider than 80 columns. You may want to open an extra-wide `xterm` window to handle this output or redirect it to a file that you can study in a text editor capable of displaying more than 80 columns. To quickly spot servers listening for connections, type **netstat -lp** rather than **netstat -ap**. The result will show all servers that are listening for connections, omitting client connections and specific server instances that are already connected to clients.

Using Remote Network Scanners

Network scanners, such as Nmap (`http://www.insecure.org/nmap/`) or Nessus (`http://www.nessus.org`), can scan for open ports on the local computer or on other computers. The more sophisticated scanners, including Nessus, will check for known vulnerabilities, so they can tell you if a server might be compromised should you decide to leave it running.

EXERCISE 7.2

Monitor Network Port Use

This exercise explores the use of netstat to monitor network port use. To do so, follow these steps:

1. Log into the Linux system as a normal user.

2. Launch an xterm from the desktop environment's menu system, if you used a GUI login method.

3. Type **netstat -ap | less** and page through the output. Chances are you'll see quite a few entries for servers that are listening for connections and for established connections to local servers or from local clients to remote servers. Pay particular attention to servers that are listening for new connections—that is, those that list LISTEN in the State column of the output.

4. Type **netstat -ap | grep ssh** to find connections involving SSH. Depending on your configuration and the servers you have running, you might see no output or many lines of output.

5. In another login session or xterm window, initiate an SSH connection to another computer. For instance, type **ssh *remote.luna.edu*** to connect to *remote.luna.edu*.

6. Type **netstat -ap | grep ssh** in your original session (that is, not in your SSH connection). Compare the output to that which you obtained in step 4. The output should have an additional line, reflecting the session you initiated in step 5.

7. Log out of the SSH session you initiated.

8. Type **netstat -ap | grep ssh** again. The output should now be missing the line for the session you've now closed.

If you're using a multi-user system, additional SSH sessions could come and go during the course of this lab, reflecting the activities of other users.

WARNING   Network scanners are used by crackers for locating likely target systems, as well as by network administrators for legitimate purposes. Many organizations have policies forbidding the use of network scanners except under specific conditions. Therefore, you should check these policies and obtain explicit permission, signed and in writing, to perform a network scan. Failure to do so could cost you your job or even result in criminal charges, even if your intentions are honorable.

Nmap is capable of performing a basic check for open ports. Pass the -sT parameter and the name of the target system to it, as shown here:

```
$ nmap -sT teela.rodsbooks.com

Starting nmap V. 3.55 ( www.insecure.org/nmap/ ) at 2004-12-21 12:11 EDT
Interesting ports on teela.rodsbooks.com (192.168.1.2):
(The 1581 ports scanned but not shown below are in state: closed)
Port        State       Service
21/tcp      open        ftp
22/tcp      open        ssh
```

> **NOTE**   As with the output of netstat shown in "Using Local Network Activity Tools," this output has been trimmed for brevity's sake.

This output shows two open ports—21 and 22, used by ftp and ssh, respectively. If you weren't aware that these ports were active, you should log into the scanned system and investigate further, using netstat or ps to locate the programs using these ports and, if desired, shut them down. The -sT option specifies a scan of TCP ports. A few servers, though, run on UDP ports, so you need to scan them by typing **nmap -sU *hostname***. (This usage requires root privileges, unlike scanning TCP ports.)

Nmap is capable of more sophisticated scans, including "stealth" scans that aren't likely to be noticed by most types of firewalls, ping scans to detect which hosts are active, and more. The Nmap man page provides details. Nessus, which is built atop Nmap, provides a GUI and a means of performing automated and still more sophisticated tests. Nessus comes as separate client and server components; the client enables you to control the server, which does the actual work.

When you use a network scanner, you should consider the fact that the ports you see from your test system may not be the same as those that might be visible to an attacker. This issue is particularly important if you're testing a system that resides behind a firewall from another system that's behind the same firewall. Your test system is likely to reveal accessible ports that would not be accessible from the outside world. On the other hand, a cracker on your local network would most likely have access similar to your own, so you shouldn't be complacent because you use a firewall. Nonetheless, firewalls can be important tools for hiding servers without shutting them down.

Uninstalling or Reconfiguring Servers

Once you've identified an unnecessary server, your task becomes one of shutting it down. Broadly speaking, two options exist:

- You can disable the server by changing its SysV configuration or disabling it in your system's super server. Consult Chapters 6 and 9 for details on how to perform these tasks. Disabling the server in this way has the advantage that you can easily reactivate the server in the future if you decide to do so. It has the disadvantage that the server's files will continue to consume disk space, and the server might be *accidentally* reactivated in the future.

- You can completely uninstall the server using your distribution's package management tools or by otherwise deleting its files. Chapter 2, "Managing Software," describes this task. Completely uninstalling software has the advantage of reducing the risk of accidental reactivation, but it has the drawback that you won't be able to easily reactivate the server intentionally should you decide to do so.

Overall, completely removing the server is generally preferable unless you merely want to temporarily disable a server. If you decide to reactivate the server in the future, you can always re-install it.

Package and Program Security

Security isn't limited to networking. Many security issues relate to local programs—program bugs, misconfiguration, and other problems can lead to security breaches. Keeping ahead of potential problems with packages and programs is therefore important to maintaining the overall security of your system. Three tasks are particularly important in this respect: tracking down (and, if necessary, reconfiguring) set user ID (SUID) and set group ID (SGID) programs, verifying the integrity of programs, and keeping programs up-to-date.

Tracking Down SUID/SGID Programs

Chapter 4, "Managing Files and Filesystems," describes the SUID and SGID bits. In brief, these are special flags that may be applied to executable program files, causing Linux to treat the program as if it were run by the program file's owner (for SUID) or by the file's group (for SGID) rather than by the individual who actually ran the program. For instance, if a program's SUID bit is set and if the program file is owned by `bruce`, the program, when run by anybody, will be able to access all the files owned by `bruce` and otherwise behave as if `bruce` had run it.

The SUID and SGID bits are frequently associated with the `root` account in order to enable them to perform tasks that require special privilege. For instance, the `passwd` program (described in Chapter 8) is SUID `root` because only `root` may modify the Linux password database. Thus, for an ordinary user to change a password, some mechanism must exist to run a process as `root`. That mechanism, in the case of `passwd`, is the SUID bit.

The problem with all of this is that the SUID and SGID bits can be security risks. For instance, suppose that the `rm` program's SUID bit was set. This program is normally owned by `root`, so setting the SUID bit on `rm` would mean that any user could delete any file on the system. Although no Linux distribution sets the SUID bit on `rm` by default, the SUID bit can be set inappropriately. This can happen by accident (say, a mistyped command by `root`), by malice (if a cracker gains access to the system), or because of a more subtle misconfiguration by the distribution maintainer (the SUID bit set unnecessarily on a program for which it's less blatantly inappropriate than `rm`). Even if the SUID or SGID bit is set appropriately, a bug in the program can become more serious because the bug executes as `root`. If the bug enables users to write files, for example, any user can exploit the bug to overwrite critical system configuration files. For these reasons, you should periodically review your system to find all the SUID programs and, if appropriate, change their configuration.

To do this, you can use the `find` command, which is described in detail in Chapter 4. In particular, you can use the `-perm` *mode* option, which searches for files with the specified permission mode. To search for SUID and SGID files, you should pass a *mode* of +6000. The symbolic representation for the SUID and SGID bits is 6000, and the plus sign (+) tells `find` to locate any file with any of the specified bits set. (You could search for SUID files alone by passing +4000 or SGID alone by passing +2000.) You might also want to pass `-type f`, which restricts the search to regular files. (Directories use the SUID and SGID bits differently, as described in Chapter 4.) Thus, to search the entire computer for SUID and SGID programs, you would type this:

```
# find / -perm +6000 -type f
```

The result will be a list of files, one per line, that have either the SUID or the SGID bits set. Programs that are likely to be present in this list include `su`, `ping`, `mount`, `passwd`, `umount`, and `sudo`. These programs all have a legitimate need to be so configured. Most systems have additional SUID and SGID programs. If you have doubts about whether the program really needs this status, you should investigate further. Try verifying the package integrity, as described in the next section, and perform a Web search on the program name and "SUID" or "SGID," as appropriate. You might also try changing the SUID status of the program using `chmod`, as described in Chapter 4, and see if it still works as it should when run by a normal user.

> **WARNING** Programs that are SUID or SGID `root` but that should not be can be a sign of system compromise. Crackers might reconfigure programs in this way in order to more easily do their dirty work. Thus, if you find such programs, investigate the overall integrity of the system. On the other hand, if a distribution maintainer set the SUID or SGID bit unnecessarily, this isn't cause for concern. Likewise, accidental misconfiguration by you or another administrator isn't cause for massive system upheaval—but you'll need to dig a bit deeper to ascertain whether such a change was accidental or a sign of a deeper problem.

Verifying Package Integrity

Crackers sometimes modify system files in order to simplify subsequent break-ins, capture other users' passwords, and otherwise do their bidding. For this reason, you should periodically review the integrity of your distribution's packages. Several tools exist to help in this task. These tools include Tripwire, `chkrootkit`, and your package manager.

Using Tripwire

Should somebody manage to break into your computer, Tripwire (`http://www.tripwire.org`) may be your best bet to detect that fact. This utility records a set of information about all the important files on a computer, including various types of *checksums* and *hashes*—short digital "signatures" that enable you to quickly determine whether or not a file has been changed. (These can also be used in other ways; for instance, Linux uses hashes to store passwords.) With this database stored in a secure location, you can check your system periodically for alteration. If an

intruder has modified any of your files, Tripwire will alert you to this fact. If you like, you can run a Tripwire verification on a regular basis—say, once a week in a `cron` job.

Many distributions ship with Tripwire, but it may not be installed by default. The utility is controlled through two configuration files: `tw.cfg` and `tw.pol`, which often reside in `/etc/tripwire`. The `tw.cfg` file controls overall configuration options, such as where `tw.pol` resides, how Tripwire sends reports to the system administrator, and so on. The `tw.pol` file includes information on the files Tripwire should monitor, among other things. Both files are binary files created from text-mode files called `twcfg.txt` and `twpol.txt`, respectively. You may need to edit `twpol.txt` to eliminate references to files that you don't have on your system and to add information on files that you do have but that the default file doesn't reference. Use the `twinstall.sh` program (which often resides in `/etc/tripwire`) to generate the binary configuration files and other critical database files. This utility will ask you to set a pair of passphrases, which are like passwords but are typically longer, to control access to the Tripwire utilities. You'll then need to enter these passphrases to have the utility do its encoding work.

Once you've generated the basic setup files, type **`tripwire --init`** to have it generate initial checksums and hashes on all the files it's configured to monitor. This process is likely to take a few minutes. Thereafter, typing **`tripwire --check`** will check the current state of the system against the database, and typing **`tripwire --update`** will update the database (say, in case you upgrade a package). The `--init` and `--update` operations require you to enter the passphrase, but `--check` doesn't. Therefore, you can include an automated Tripwire check in a `cron` job. (Chapter 8 describes `cron` jobs in more detail.)

> **NOTE** Tripwire is best installed and initialized on a completely fresh system, before connecting the computer to the Internet but after all programs have been configured. Although it's possible to install it on a system that's been up and running for some time, if that computer has already been compromised without your knowledge, Tripwire won't detect that fact.

Using *chkrootkit*

The `chkrootkit` program (`http://www.chkrootkit.org`) is something of a last-resort method of detecting intrusion and is the closest thing in the Linux world to Windows virus scanners. (Linux virus scanning programs also exist, but they're intended mainly to check for Windows viruses on Samba shares. Linux viruses are not a problem in the real world, at least not as of mid-2005.)

Many crackers use *root kits,* which are prepackaged intrusion tools. When an intruder runs a root kit against a target, the root kit software probes for known weaknesses (such as servers with known security bugs), breaks in, and installs software to enable simpler access by the intruder. The intruder can then log in using Telnet, SSH, or the like and gain full control of the system.

> **NOTE** Intruders who use root kits are often referred to as *script kiddies.* These miscreants have minimal skill; they rely on the root kit to do the real work of the intrusion. Some people prefer to reserve the term *cracker* for more skilled intruders, but others consider script kiddies to be crackers with minimal expertise.

Using `chkrootkit` is fairly straightforward: Type its name. The result is a series of lines summarizing checks that the software performs. These lines should all end with `not infected`, `no suspect files found` or a similar reassuring messages. If any message alerts you to an intrusion, you should take immediate corrective measures.

Using Package Manager Checksums

Package managers—most notably the RPM Package Manager (RPM)—maintain checksums on all their installed packages. As such, they can be used as intrusion detection tools. In particular, the `-V` (or `--verify`) option to `rpm` performs a package verification:

```
# rpm -V postfix
S.5....T c /etc/postfix/main.cf
S.5....T c /etc/postfix/sasl_passwd
S.5....T c /etc/postfix/sender_canonical
```

Each line of output reports files that have changed in some way. The first eight characters of the output lines report what's changed: the file size, the mode, the MD5 sum, device major or minor numbers (for device files), link path mismatch, user ownership, group ownership, or time. A dot (`.`) in a position indicates that a test passed; any other character denotes a change (the character used depends on the test and is intended to be mnemonic). Following the eight-character block may be another character that denotes the file type—`c` for configuration files, `d` for documentation, `g` for "ghost" files (those not included in the actual package), `l` for a license file, or `r` for a README file. Files that haven't changed are *not* displayed in the output.

In the preceding example, three files have changed: `main.cf`, `sasl_passwd`, and `sender_canonical`. All three files are marked as configuration files (the `c` characters preceding the filenames), and all three have changed file sizes, MD5 sums, and times. Because these are configuration files, these changes aren't particularly suspicious, but a similar pattern of changes in program executables would be cause for concern. Changes to the MD5 sum (the form of checksum used by RPM) are particularly likely to be the result of tampering; they indicate that the file's contents have changed compared to the file in the original package. Time stamp changes can sometimes be completely innocent. Ownership and permissions changes might be the result of unwanted tampering or could be innocent in some cases (say, if you've deliberately removed world execute permission to improve security).

You can verify an individual package by providing a package name, as in the preceding example. You can also verify all the packages installed on a system by passing the `-a` option. The result is likely to be a very long list, though, because so many packages include configuration files and other ancillary files that are normally changed. You might want to pass the output to a file that you can peruse later, as in **rpm -Va > rpm-test.txt**.

WARNING   One major limitation of a package manager's checksums for detecting intruders is that it's very easily overcome. Unlike Tripwire's database, the RPM database isn't password-protected. In fact, intruders can easily cover their tracks by using RPM itself to install their modified tools. When you attempt to use RPM to verify the package, RPM will merrily report no problems.

Keeping Packages Up-to-Date

Program bugs are discovered every day. Some of these bugs have security implications. Sometimes the bug is in a server and enables remote users to do things they shouldn't be able to do; other times the bug is in a local program and can be exploited by local users to acquire privileges they should not have. In either case, upgrading programs with such bugs in a timely manner is extremely important, particularly on server systems and multi-user computers. Fortunately, several resources exist to help keep you abreast of problems, and many distributions provide tools that can be helpful in this regard.

Resources to Review for Update Information

You should get into the habit of reviewing several security websites and other resources to learn about new threats:

CERT/CC   The Computer Emergency Response Team Coordination Center (`http://www.cert.org`) hosts general security information, including information on the latest threats. Periodically reviewing this site will help you keep up-to-date with security developments.

US-CERT   The United States Computer Emergency Readiness Team (`http://www.us-cert.gov`) has taken over some of the duties formerly held by CERT/CC. In practice, both sites are worth monitoring.

CIAC   The Computer Incident Advisory Capability (`http://ciac.llnl.gov/ciac/`), run by the U.S. Department of Energy, is similar to CERT/CC and US-CERT in many respects, but its web page gives greater emphasis to current threats and less coverage of general security practices.

CVE   The Computer Vulnerabilities and Exposures (`http://cve.mitre.org`) site is dedicated to maintaining a dictionary of vulnerability names. This information can be useful in facilitating communication about problems. The CVE contains less in the way of descriptions of the vulnerabilities and exploits it names, though.

SecurityFocus and Bugtraq   The SecurityFocus website (`http://www.securityfocus.com`) is yet another general security site. One of its important features is that it hosts the Bugtraq mailing list (`http://www.securityfocus.com/archive/1`), which can be a good way to keep informed—subscribe, and alerts about new threats will be delivered to your e-mail account soon after they're made public.

Linux Security   The Linux Security site, `http://www.linuxsecurity.com`, is very similar to CERT/CC, US-CERT, CIAC, and SecurityFocus in many ways. Linux Security, though, caters to Linux in particular and so may be more helpful in addressing Linux-specific issues or in pointing to Linux-specific fixes.

Distributions' websites   Most Linux distributions maintain security information on their websites. Go to your distribution's main page and look for links relating to security. These sites can provide specific upgrade instructions for your distribution in particular.

Product web pages and mailing lists   Many programs have web pages and mailing lists, and these can be good resources for learning of security problems related to these programs. Of course, regularly perusing all of the pages related to the hundreds of programs that make up a Linux system can be a full-time job. You might want to keep an eye on the web pages or mailing lists for any high-profile server programs that you run, such as the Apache web server or the sendmail mail server.

Security newsgroups   Several Usenet newsgroups are devoted to security. Of particular interest are the groups in the `comp.security` hierarchy.

I recommend you investigate most or all of these resources and then keep up with a few of them. For instance, you might check the CERT/CC and Linux Security websites on a daily basis, subscribe to the Bugtraq mailing list, and check your distribution's security page on a weekly basis. Keeping up with security developments in this way will alert you to potential problems quickly—with any luck, quickly enough to avoid problems caused by crackers who might try to exploit weaknesses soon after they're discovered.

> Many of these resources offer RDF Site Summary (RSS) feeds of their content. This protocol enables you to use a news aggregator program, such as AmphetaDesk (`http://www.disobey.com/amphetadesk/`) or BlogBridge (`http://www.blogbridge.com`), to track security problems and learn about them as soon as possible.

Tools to Help Update Your System

Most distributions provide tools that can help you keep your system up-to-date. In particular, your package manager (RPM or Debian) probably supports tools that will enable you to easily download and install updates to your system software soon after they become available from your distribution maintainer. For instance, the Advanced Packaging Tool (APT) described in Chapter 2 enables you to upgrade all your software by typing two commands:

```
# apt-get update
# apt-get dist-upgrade
```

> Substituting upgrade for `dist-upgrade` performs a system upgrade in a somewhat more conservative way, which can reduce the risk of running into dependency problems.

Although APT originated as a Debian tool, it's been ported to work with RPM-based distributions, as described in Chapter 2. Most modern RPM-based distributions provide their own tools, such as Fedora and Red Hat's Update Agent and SuSE's YaST. These tools are highly distribution-centric, so you should consult the distribution's documentation for details of their use. Most of these tools are X-based programs that are aimed at desktop users, but

you can certainly use them on servers if you run X on the server (even just for an update session) or if you log in from a remote system that's running X.

Because APT is a text-based tool, you can call it in a `cron` job to have it run on a regular basis. (Chapter 8 describes `cron` jobs in more detail.) This is reasonably safe for upgrading the database (via the `update` option to `apt-get`), but I recommend against automatically updating the system (via `upgrade` or `dist-upgrade`). Doing so creates the potential for upgrades to go wrong when you're not around to fix the problem. You could, though, run these options in conjunction with `-s`, which simulates the action. If done in a `cron` job with proper e-mail redirection, the result will be an e-mailed report of the packages for which updates are available being sent to you on a regular basis. You can then decide whether or not to upgrade the packages.

Passwords

A default Linux configuration relies heavily on passwords. Users' passwords are their keys into the system, and if users are careless with their passwords, security breaches can result. Understanding these risks is critical to maintaining system security, but this is one task for which you *must* enlist the help of your users; after all, they're the ones who are in possession of their passwords! You should also be aware of some of the tools Linux provides to help keep passwords secure. (Most of the details concerning password-related commands are described in Chapter 8.)

Password Risks

Passwords can end up in crackers' hands in various ways, and you must take steps to minimize these risks. Steps you can take to improve your system's security include the following:

Use strong passwords.   Users should employ good passwords, as described shortly, in "Choosing a Good Password." This practice won't eliminate all risk, though.

Change passwords frequently.   You can minimize the chance of damage due to a compromised password by changing passwords frequently. Some Linux tools can help to enforce such changes, as described briefly in "Tools for Managing Passwords" and in more detail in Chapter 8.

Use shadow passwords.   If a cracker who's broken into your system through an ordinary user account can read the password file, or if one of your regular users is a cracker who has access to the password file, that individual can run any of several password-cracking programs on the file. For this reason, you should use shadow passwords stored in `/etc/shadow` whenever possible. Most Linux distributions use shadow passwords by default. If yours doesn't, consult the upcoming section "Tools for Managing Passwords" for information on enabling this feature.

Keep passwords secret.   You should remind your users not to reveal their passwords to others. Such trust is sometimes misplaced, and sometimes even a well-intentioned password recipient might slip up and let the password fall into the wrong hands. This can happen by writing the password down, storing it in electronic form, or sending it by e-mail or other electronic means. Indeed, users shouldn't e-mail their own passwords even to themselves because e-mail can be intercepted.

Use secure remote login protocols.   Certain remote login protocols are inherently insecure; all data traverse the network in an unencrypted form. Intervening computers can be configured to snatch passwords from such sessions. Because of this, it's best to disable Telnet, File Transfer Protocol (FTP), and other protocols that use cleartext passwords in favor of protocols that encrypt passwords, such as SSH.

Be alert to shoulder surfing.   If your users log in using public terminals, as is common on college campuses, in Internet cafes, and the like, it's possible that others will be able to watch them type their passwords (a practice sometimes called "shoulder surfing"). Users should be alert to this possibility and minimize such logins if possible.

Use each password on just one system.   If one computer's password database is compromised, and if users of that system reuse their passwords on other systems, those other systems can be compromised. For this reason, it's best to use each password just once. Unfortunately, the proliferation of websites that require passwords for access makes this rule almost impossible to enforce, at least without violating the rule of not writing the password down. (Modern web browsers can remember passwords for you, but this is done by storing them in a file—essentially, writing them down.) A reasonable compromise might be to use one password for the least-sensitive websites (such as online newspapers) and unique passwords for sensitive websites (such as banking sites) and login accounts.

Be alert to social engineering.   Crackers often use *social engineering* to obtain passwords. This practice involves tricking individuals into giving up their passwords by pretending to be a system administrator or by otherwise misleading victims. Amazingly, a large percentage of people fall for this ploy. A related practice is *phishing,* in which an attacker puts up a fake website or sends an e-mail pretending to be from somebody else. The victim is then lured into revealing sensitive data (often credit card numbers).

Some of these steps are things you can do, such as replacing insecure remote login protocols with encrypted ones. Others are things your users must do. This illustrates the importance of user education, particularly on systems with many users.

Choosing a Good Password

As a general rule, people tend to be lazy when it comes to security. In computer terms, this means that users tend to pick passwords that are easy to guess, and they change them infrequently. Both these conditions make a cracker's life easier, particularly if the cracker knows the victim. Fortunately, Linux includes tools to help make your users select good passwords and change them regularly.

Common (and therefore poor) passwords include those based on the names of family members, friends, and pets; favorite books, movies, television shows, or the characters in any of these; telephone numbers, street addresses, or Social Security numbers; or other meaningful personal information. Any single word that's found in a dictionary (in *any* language) is a poor choice for a password. The best possible passwords are random collections of letters, digits, and punctuation. Unfortunately, such passwords are difficult to remember. A reasonable compromise is to build a password in two steps: First, choose a base that's easy to remember but difficult to guess. Second, modify that base in ways that increase the difficulty of guessing the password.

One approach to building a base is to use two *unrelated* words, such as *bun* and *pen*. You can then merge these two words (bunpen). Another approach, and one that's arguably better than the first, is to use the first letters of a phrase that's meaningful to the user. For instance, the first letters of "yesterday I went to the dentist" become yiwttd. In both cases, the base should not be a word in any language. As a general rule, the longer the password the better. Older versions of Linux had password length limits of eight characters, but those limits have been lifted by the use of the MD5 password hash, which is the standard on modern Linux distributions. Many Linux systems require passwords to be at least four to six characters in length; the passwd utility won't accept anything shorter than the distribution's minimum.

With the base in hand, it's time to modify it to create a password. The user should apply at least a couple of several possible modifications:

Adding numbers or punctuation   The single most important modification is to insert random numbers or punctuation in the base. This step might yield, for instance, bu3npe&n or y#i9wttd. As a general rule, add at least two symbols or numbers.

Mixing case   Linux uses case-sensitive passwords, so jumbling the case of letters can improve security. Applying this rule might produce Bu3nPE&n and y#i9WttD, for instance.

Order reversal   A change that's very weak by itself but that can add somewhat to security when used in conjunction with the others is to reverse the order of some or all letters. You might apply this to just one word of a two-word base. This could yield Bu3nn&EP and DttW9i#y, for instance.

Your best tool for getting users to pick good passwords is to educate them. Tell them that passwords can be guessed by malicious individuals who know them or even who target them and look up personal information in telephone books, on web pages, and so on. Tell them that, although Linux encrypts its passwords internally, programs exist that feed entire dictionaries through Linux's password encryption algorithms for comparison to encrypted passwords. If a match is found, the cracker has found the password. Therefore, using a password that's not in a dictionary, and that isn't a simple variant of a dictionary word, improves security substantially. Tell your users that their accounts might be used as a first step toward compromising the entire computer or as a launching point for attacks on other computers. Explain to your users that they should *never* reveal their passwords to others, even people claiming to be system administrators—this is a common scam, but real system administrators don't need users' passwords. You should also warn them not to use the same password on multiple systems because doing so quickly turns a compromised account on one system into a compromised account on all the systems. Telling your users these things will help them understand the reasons for your concern, and it is likely to help motivate at least some of them to pick good passwords.

If your users are unconcerned after being told these things (and in any large installation, some will be), you'll have to rely on the checks possible in passwd. Most distributions' implementations of this utility require a minimum password length (typically four to six characters). They also usually check the password against a dictionary, thus weeding out some of the absolute worst passwords. Some require that a password contain at least one or two digits or punctuation.

WARNING

Password-cracking programs, such as Crack (`http://www.crypticide.org/users/alecm/`), are easy to obtain. You might consider running such programs on your own encrypted password database to spot poor passwords, and in fact, this is a good policy in many cases. It's also grounds for dismissal in many organizations and can even result in criminal charges being brought, at least if done without authorization. If you want to weed out bad passwords in this way, discuss the matter with your superiors and obtain written permission from a person with the authority to grant it before proceeding. Take extreme care with the files involved, too; it's best to crack the passwords on a computer with *no* network connections.

Another password security issue is password changes. Frequently changing passwords minimizes the window of opportunity for crackers to do damage; if a cracker obtains a password but it changes before the cracker can use it (or before the cracker can do further damage using the compromised account), the password change has averted disaster. As described shortly, you can configure accounts to require periodic password changes. When so configured, an account will stop accepting logins after a time if the password isn't changed periodically. (You can configure the system to warn users when this time is approaching.) This is a very good option to enable on sensitive systems or those with many users. Don't set the expire time too low, though—if users have to change their passwords too frequently, they'll probably just switch between a couple of passwords, or pick poor ones. Precisely what "too low" a password change time is depends on the environment. For most systems, one to four months is probably a reasonable change time, but for some it might be longer or shorter.

Tools for Managing Passwords

Most Linux distributions use shadow passwords by default, and for the most part, this chapter is written with the assumption that this feature is active. In addition to providing extra security by moving hashed passwords out of the world-readable `/etc/passwd` file and into the more secure `/etc/shadow` file, shadow passwords add extra account information.

If you happen to be using a system that hasn't enabled shadow passwords but you want to add that support, you can do so. Specifically, the `pwconv` and `grpconv` programs do this job for `/etc/passwd` and `/etc/group`, respectively. These programs take no parameters; simply type their names to create `/etc/shadow` and `/etc/gshadow` files that hold the passwords and related account control information, based on the contents of the unprotected files. You can run these programs even if most accounts already use shadow passwords; when used in this way, any accounts that do not yet use shadow passwords are converted, and those that already exist are left untouched. This feature can be handy when duplicating accounts from one system on another—you can cut and paste `/etc/passwd` entries from an old system that doesn't use shadow passwords and then type **pwconv** to create appropriate shadow password entries.

> **WARNING** The pwconv and grpconv utilities can loop forever or otherwise misbehave if their source files are malformed. You may want to check the format of these files with pwck and grpck. These programs look for errors such as an incorrect number of fields, missing home directories, and duplicate names, and report any problems to you.

Although converting a system so that it no longer uses shadow passwords is generally inadvisable, you can do so with the pwunconv and grpunconv commands, which merge the contents from the shadow files back into /etc/passwd and /etc/group, respectively. In practice, you're most likely to need to do this if you must move Linux account information over to a computer that doesn't support shadow passwords. If you do this, you may want to first back up the /etc/passwd and /etc/shadow (or /etc/group and /etc/gshadow) files. You can then copy the backups back to /etc when you're done, reversing the process. This procedure will preserve some shadow information, such as account expiration data, that will be lost in the conversion process.

One of the advantages of shadow passwords is that they support password aging and account expiration features. These features enable you to enforce password changes at regular intervals or to automatically disable an account after a specified period of time. You can enable these features and set the times using the chage command, which is described in more detail in Chapter 8.

Configuring User-Level Security

Several security measures apply to individual users. These tools can help improve system security by reducing the need to log in under sensitive usernames (such as root) or by limiting the system resources that a single individual may consume.

Configuring Mail Aliases

Most Linux systems (even desktop computers) run mail servers. If nothing else, these server programs handle local mail delivery. Such deliveries can originate from automatic processes. For instance, the cron utility (described in Chapter 8) can send e-mail reports of its activities to the user who runs cron jobs. In many cases, such e-mail is sent to the root account, which can pose a problem: The e-mail will either go unnoticed and unread or somebody will have to log in as root to read the mail. Reading mail as root is a security no-no because the power of root should be used sparingly lest a typo wipe out the system. Thus, it's desirable to redirect mail to root (and perhaps other system accounts, if any receive mail) to another user.

You can redirect e-mail by creating a mail aliases file. This file is usually called aliases and is located in /etc or a subdirectory of /etc devoted to your mail server, such as /etc/postfix for the Postfix mail server or /etc/mail for sendmail. This file's format is fairly straightforward: It consists of lines, each of which is either a comment (denoted by a hash

mark, #) or an alias line. Alias lines consist of a name to be aliased, a colon, and a comma-separated list of addresses to which the mail should be redirected:

```
postmaster: root
root: benf
```

This example sets up two aliases: Mail sent to `postmaster` is redirected to `root`, and mail sent to `root` is redirected to `benf`. This example illustrates the fact that mail can be redirected more than once. In this example, mail sent to `postmaster` will ultimately be sent to `benf`.

The addresses (such as `benf`) can be local accounts, local filenames, commands preceded by a pipe (such as `|processor` to send the message through `processor`), or remote e-mail addresses (such as `benf@pangaea.edu`).

Once you've configured this file, it must be converted to binary format (typically stored in `aliases.db`) by typing **newaliases**. If you fail to do this, your changes will have no effect.

This aliasing system works with major Linux mail servers, such as sendmail and Postfix. If you're using a particularly exotic mail server, you should consult its documentation to see if it supports this mechanism or uses something else.

Setting Login, Process, and Memory Limits

Sometimes you may want to impose limits on how many times users may log in, how much CPU time they can consume, how much memory they can use, and so on. Imposing such limits is best done through a Pluggable Authentication Modules (PAM) module called `pam_limits`. Most major Linux distributions use this module as part of their standard PAM configuration, so chances are you won't need to add it; however, you will still need to configure `pam_limits`. This is done by editing its configuration file, /etc/security/limits.conf. This file contains comments (denoted by a hash mark, #) and limit lines that consist of four fields:

domain type item value

Each of these fields specifies a particular type of information:

The domain   The *domain* describes the entity to which the limit applies. It can be a user-name; a group name, which takes the form *@groupname*; or an asterisk (*) wildcard, which matches everybody.

Hard or soft limits   The *type* field specifies the limit as `hard` or `soft`. A `hard` limit is imposed by the system administrator and cannot be exceeded under any circumstances, whereas a `soft` limit may be temporarily exceeded by a user. You can also use a dash (-) to signify a limit is both `hard` and `soft`.

The limited item   The *item* field specifies what type of item is being limited. Examples include `core` (the size of core files), `data` (the size of a program's data area), `fsize` (the size of files created by the user), `nofile` (the number of open data files), `rss` (the resident set size), `stack` (the stack size), `cpu` (the CPU time of a single process in minutes), `nproc` (the number of concurrent processes), `maxlogins` (the number of simultaneous logins), and `priority` (the process priority). The

data, rss, and stack items all relate to memory consumed by a program. These and other measures of data capacity are measured in kilobytes.

The value   The final field specifies the value that's to be applied to the limit.

As an example, consider a system on which certain users should be able to log in and perform a limited number of actions but not stay logged in indefinitely and consume vast amounts of CPU time. You might use a configuration like this one:

```
@limited  hard  cpu  2
```

This configuration applies a hard CPU limit of two minutes to the limited group. Members of this group will be able to log in and run programs, but if one of those programs consumes more than two minutes of CPU time, it will be terminated.

> CPU time and total system access time are two entirely different things. CPU time is calculated based on the amount of time that the CPU is actively processing a user's data. Idle time (for instance, when a user's shell is active but no CPU-intensive tasks are running) doesn't count. Thus, a user can log in and remain logged in for hours even with a very low hard CPU time limit. This limit is intended to prevent problems caused by users who run very CPU-intensive programs on systems that should not be used for such purposes.

One particularly radical approach to security is to use the /etc/nologin file. If this file is present, only root may log into the computer. In many respects, this is like setting critical system limits to 0 for all other users. This file is most likely to be useful on dedicated server systems that have no regular console or remote shell users.

Summary

Linux documentation and information resources are varied. The traditional Unix documentation system has long been the man page, and Linux has adopted this system; but other documentation tools coexist with man pages. Some projects have shifted their emphasis to the competing info tool. Most projects also provide additional documentation that doesn't fit in the man (or info) page format, such as tutorial documents. You can often find such documentation on your system or on the Internet, either on the package's main web page or at the LDP site.

Maintaining system security is both important and time-consuming. A great deal of security emphasis is on network security, and for this, configuring firewalls and your super server will go a long way by restricting access to specific ports. You should also disable unused servers, verify package integrity, and keep your packages up-to-date. You must know where to look to find the latest security information (several websites and other Internet resources are helpful, such as the CERT/CC and CIAC sites and various Usenet newsgroups). Attending to passwords and performing miscellaneous tasks to keep your local accounts from becoming security risks are also important security tasks.

Exam Essentials

Summarize commands for obtaining help in Linux. The `man` and `info` commands both display summary information on how to use commands, configuration files, system calls, and so on. The `whatis` and `apropos` commands both search the `man` database for the keyword you specify, the `apropos` command searching more fields than `whatis`.

Describe where packages store their documentation. Aside from `man` pages and in-program help files, most Linux packages store documentation in the `/usr/doc`, `/usr/share/doc`, `/usr/X11R6/doc`, or similar directories. This documentation may consist of `README` and similar files from the source code distribution, formatted manuals, and other miscellaneous documentation.

Summarize the three main types of documentation maintained by the LDP. The Linux Documentation Project (LDP) hosts HOWTOs, Guides, and FAQs. HOWTOs are task-oriented introductions to programs or topics. Guides are longer (often book-length) tutorial documents. FAQs provide quick answers to common questions.

Describe some of the tools you can use to deliver messages to text-mode login users. The `/etc/issue` and `/etc/issue.net` files hold messages that are displayed prior to the `login:` prompt on the console and by Telnet, respectively. The `/etc/motd` file holds the message of the day, which is displayed immediately after login. The `shutdown` command enables you to deliver a message to users as a scheduled shutdown time approaches, warning them of the impending event.

Explain the function of a firewall. A firewall blocks access to specific ports on a computer, which can be a router (thus protecting an entire network) or an individual server or workstation. Firewalls block all access to specific ports no matter what program might be using those ports.

Summarize Linux firewall tools. Modern Linux systems use the `iptables` program to implement firewall rules. Typically, a script calls `iptables` many times, once for each rule you want to implement. Older Linux systems used the `ipchains` or `ipfwadm` programs in a similar way, although details differ. (Modern Linux systems can still use these tools to implement legacy firewall scripts.)

Explain the function of super server port access controls. Super servers or programs called by them (such as TCP Wrappers) can restrict access to ports for the servers they manage. These restrictions occur at a higher level than a firewall's restrictions, and they apply only to the servers managed by the super server.

Describe why SUID and SGID programs are potentially risky. The set user ID (SUID) and set group ID (SGID) bits tell Linux to run the program as the user or group that owns the file. This is particularly risky when `root` owns the program file because it essentially elevates all users to `root` for the purposes of running the file, making bugs in the program more dangerous and raising the possibility of a clever user abusing the program to acquire full `root` privileges or otherwise wreaking havoc.

Summarize methods of verifying package integrity. The most thorough way of checking package integrity is via the Tripwire program, which was designed as a secure method of checking that

files haven't been modified. Linux package systems can do the same, but they weren't designed with security in mind and so aren't as trustworthy in this role. The chkrootkit program can scan a system for known suspicious files, much like a Windows virus scanner.

Explain how to generate a good password.   Ideally, passwords should be random. Failing that, one good approach is to generate a base that's hard to guess and then modify it by adding digits and punctuation, changing the case of some characters, and changing letter order.

Review Questions

1. Which of the following statements is a fair comparison of HOWTO documents to man pages?

   A. HOWTOs do not require Internet access to be read; man pages do.

   B. HOWTOs are a type of electronic documentation; man pages are printed.

   C. HOWTOs are programmers' documents; man pages describe software from a user's point of view.

   D. HOWTOs are tutorial in nature; man pages are briefer reference documents.

2. Which of the following commands searches the man page database for entries or descriptions that include a specified keyword?

   A. info

   B. apropos

   C. grep

   D. apackage

3. What file would you edit to change the man path?

   A. /etc/man.conf

   B. /etc/conf.man

   C. /etc/paths.conf

   D. /etc/conf.paths

4. Which of the following is a difference between man pages and info pages?

   A. man pages are designed to be translated by machine translation programs for reading in over a dozen languages; info pages lack this feature.

   B. man pages are the preferred documentation system of the Free Software Foundation; the FSF shuns the info page system.

   C. man pages are "flat" text files, much like ASCII text files; info pages are structured with hyperlinks, much like HTML documents.

   D. man pages describe only system calls and other programming tools; info pages describe only user programs and configuration files.

5. What information is certain to appear in a package's /usr/share/doc subdirectory?

   A. A PDF reference manual.

   B. An HTML reference manual.

   C. A sample configuration file.

   D. None of the above is certain to appear.

6. What are the three main classes of documents hosted by the Linux Documentation Project?

    A. RFCs

    B. HOWTOs

    C. FAQs

    D. Guides

7. What do you need to read Usenet newsgroups? (Select all that apply.)

    A. A newsreader

    B. A password to the Reuters website

    C. Access to a news server

    D. The Usenet network stack in your kernel

8. You recently "chatted" with a group of people online to help fix a problem in real-time with the PatheticMail program. What type of Internet help resource did you use?

    A. A web forum

    B. IRC

    C. Usenet news

    D. A mailing list

9. You want to deliver a message to would-be Telnet users that your system is for official use by employees of your company. What file would you edit to deliver this message before Telnet users see a `login:` prompt?

    A. `/etc/profile.net`

    B. `/etc/profile`

    C. `/etc/issue.net`

    D. `/etc/issue`

10. In reviewing files in `/etc`, you find that `/etc/motd` is completely empty. What does this mean?

    A. The system won't display a message of the day after users log in.

    B. Morton's Own Temporary Documentation is not installed on the system.

    C. The `motd.conf` file's name has become corrupted and its contents lost.

    D. Users will see no witty aphorisms appear on the screen after logging in.

11. Which of the following are firewall tools that may be used with 2.6.x kernels? (Select all that apply.)

    A. `ipfwadm`

    B. `ipfire`

    C. `ipchains`

    D. `iptables`

12. A server/computer combination appears in both `hosts.allow` and `hosts.deny`. What's the result of this configuration when TCP Wrappers runs?

    A. TCP Wrappers refuses to run and logs an error in `/var/log/messages`.

    B. The system's administrator is paged to decide whether to allow access.

    C. `hosts.deny` takes precedence; the client is denied access to the server.

    D. `hosts.allow` takes precedence; the client is granted access to the server.

13. When is the `bind` option of `xinetd` most useful?

    A. When you want to run two servers on one port

    B. When you want to specify computers by name rather than IP address

    C. When `xinetd` is running on a system with two network interfaces

    D. When resolving conflicts between different servers

14. What is the best way to remain abreast of security developments?

    A. Read the CERT/CC website on a regular basis.

    B. Subscribe to and read the Bugtraq mailing list.

    C. Check your distribution's security web page on a regular basis.

    D. All of the above

15. At what point during system installation should you configure Tripwire?

    A. Prior to installing major servers like Apache

    B. After installing major servers but before configuring them

    C. After installing and configuring major servers but before connecting the computer to the Internet

    D. After connecting the computer to the Internet and running it for one to four weeks

16. Of the following, which is the best password?

    A. `Odysseus`

    B. `iA710ci^My`

    C. `pickettomato`

    D. `Denver2Colorado`

17. Which of the following types of attacks involves sending bogus e-mail to lure unsuspecting individuals into divulging sensitive financial or other information?

    A. Phishing

    B. Script kiddies

    C. Spoofing

    D. Ensnaring

18. You want to configure a system so that e-mail addressed to `root` is sent to `hires@sassafras`
`.example.com`. Which file might you edit to accomplish this goal?

   A. `/etc/mail/forwarding-rules`

   B. `/var/spool/mail/root`

   C. `/etc/aliases`

   D. None of the above

19. You've discovered that your system is *not* using shadow passwords. What action should
you take?

   A. Browbeat your users to get them to use shadow passwords.

   B. Run the Crack program to verify that your passwords are still good.

   C. Run the `shadow-suite` program to generate shadow password files.

   D. Run the `pwconv` and `grpconv` programs to generate shadow password files.

20. Your login server is using PAM and you want to limit users' access to system resources. Which
configuration file will you need to edit?

   A. `/etc/limits.conf`

   B. `/etc/pam/limits.conf`

   C. `/etc/security/limits.conf`

   D. `/etc/security/pam/limits.conf`

Answers to Review Questions

1. D. HOWTOs are intended as introductions to packages or broad topics. On the other hand, `man` pages are intended to give you quick information on commands, configuration files, or the like.

2. B. The `apropos` utility searches the `man` page database for entries or descriptions that include a specified keyword. The `info` utility simply returns help information on a utility, while `grep` is an all-purpose searching tool not focused on `man` pages. There is no standard utility called `apackage`.

3. A. The `man` configuration file is `/etc/man.conf`. Among other things, this file sets the `man` path. Options B, C, and D all refer to fictitious files.

4. C. The `man` page system provides no hyperlinks; individual documents are "flat." The `info` page system, by contrast, uses hyperlinks to help organize its documents. In principle, you can pass either `man` pages or `info` pages through machine translation software, but neither system was designed with this use in mind. Option B is essentially backwards; the FSF prefers `info` pages to `man` pages. Both `man` pages and `info` pages describe system calls, programming tools, user programs, and configuration files; Option D describes a nonexistent distinction.

5. D. Although options A, B, and C are all *possible* contents of a package's documentation directory, none of these is *certain* to appear. These directories' contents are inherently uncertain, but they often contain whatever documentation doesn't fit elsewhere on the system.

6. B, C, D. The LDP hosts three main classes of documents: HOWTOs, which are tutorial documents designed to help you learn how to use a program or perform a task; FAQs, which are collections of common questions and answers about Linux; and guides, which are book-length tutorial and reference documents. RFCs are Requests for Comments, which are networking standards documents that are not maintained by the LDP.

7. A, C. Usenet newsgroups are distributed via a global collection of news servers, which exchange messages with one another. To read Usenet news, you need a newsreader program and access to one of these news servers, which are run by ISPs, universities, and businesses. Usenet news has nothing to do with the Reuters news service or its website, as option B suggests. Usenet is also not a network stack, as option D suggests.

8. B. Internet Relay Chat (IRC) is a real-time system for communicating with a group of people. It can be used to get real-time help in solving problems. Web forums, Usenet news, and mailing lists can all be mediums for obtaining help, and often fairly quickly, but not in real time.

9. C. The `/etc/issue.net` file contains text that's sent by the Telnet server to the remote Telnet client prior to Telnet's sending the `login:` prompt—precisely what the question requires. There is no standard `/etc/profile.net` configuration file, and `/etc/profile` is a configuration file for the `bash` shell, which won't help achieve the question's goals. The `/etc/issue` file is similar to `/etc/issue.net`, but it controls the message that's displayed on the local text-mode console.

10. A. The `/etc/motd` file holds the message of the day, which is displayed after users log in using text-mode tools such as text-mode consoles or SSH. An empty `/etc/motd` file simply means that no such message will appear, as option A describes. There is no such thing as Morton's Own Temporary Documentation, referred to in option B. There is no standard `/etc/motd.conf` file, as described in option C. Witty aphorisms are created by the `fortune` program, which is usually called in ways that don't involve `/etc/motd`. Although they might not be displayed, as option D suggests, there's no linkage between this and the empty `/etc/motd` file; fortunes are enabled or disabled in other ways.

11. A, C, D. The `ipfwadm` and `ipchains` programs were the preferred firewall tools for pre-2.4.x Linux kernels, but they remain as options up to at least the 2.6.x kernel series. The `iptables` program was introduced along with the 2.4.x kernel and is the preferred firewall tool for use with the 2.6.x kernel. There is no standard `ipfire` program.

12. D. TCP Wrappers uses this feature to allow you to override broad denials by adding more specific explicit access permissions to `hosts.allow`, as when setting a default deny policy (`ALL : ALL`) in `hosts.deny`.

13. C. The `bind` option of `xinetd` lets you tie a server to just one network interface rather than link to them all. It has nothing to do with running multiple servers on one port, specifying computers by hostname, or resolving conflicts between servers.

14. D. To obtain the best possible view of security developments, you should consult as many sources as possible. The CERT/CC website, the Bugtraq mailing list, and your distribution's security page are three good sources of information (but by no means the only three available). Consulting just one of these sources may not be adequate because an obscure issue that's important to you might escape notice on a single source.

15. C. Tripwire records checksums and hashes of major files, including server executables and con-figuration files. Thus, these files should be in place and properly configured before you configure Tripwire. Once the system has been running on the Internet, there's a chance that it's been com-promised; you should install Tripwire prior to connecting the computer to the Internet in order to reduce the risk that its database reflects an already-compromised system.

16. B. Ideally, passwords should be completely random but still memorable. Option B's password was generated from a personally meaningful acronym and then modified to change the case of some letters and add random numbers and symbols. This creates a password that's close to ran-dom but still memorable. Option A uses a well-known mythological figure, who is likely to be in a dictionary. Option C uses two common words, which is arguably better than option A, but not by much. Option D uses two closely related words separated by a single number, which is also a poor choice for a password.

17. A. Phishing involves sending bogus e-mail or setting up fake websites that lure unsuspecting individuals into divulging sensitive financial or other information. Script kiddies are intruders who use root kits. Spoofing involves pretending data is coming from one computer when it is coming from another. Ensnaring is not a type of attack.

18. C. The `aliases` file, commonly found in `/etc`, `/etc/mail`, or a server-specific subdirectory of `/etc`, holds mail aliases—addresses or other forwarding rules to use for accounts whose mail should be permanently forwarded elsewhere. There is no standard `/etc/mail/forwarding-rules` file. Although `/var/spool/mail/root` may exist (it's the `root` mail spool file on many systems), you wouldn't edit it to adjust forwarding for the `root` (or any other) account.

19. D. The `pwconv` and `grpconv` programs convert the `/etc/passwd` and `/etc/group` files, respectively, into forms that use shadow password files (`/etc/shadow` and `/etc/gshadow`, respectively). Because shadow passwords are desirable, taking this action is also desirable. Users have no control over whether the system uses shadow passwords, so option A will be unproductive. The Crack program attempts to discover poorly chosen passwords, but such passwords can exist on either shadowed or non-shadowed systems, so option B won't affect the system's shadow password status. There is no standard `shadow-suite` program, as referenced in option C.

20. C. The `/etc/security/limits.conf` file holds the configuration settings that will allow you to limit users' access. The other options listed do not give the correct path to this file.

Administering the System

THE FOLLOWING LINUX PROFESSIONAL INSTITUTE OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **1.111.1 Manage users and group accounts and related system files (weight: 4)**
- ✓ **1.111.2 Tune the user environment and system environment variables (weight: 3)**
- ✓ **1.111.3 Configure and use system log files to meet administrative and security needs (weight: 3)**
- ✓ **1.111.4 Automate system administration tasks by scheduling jobs to run in the future (weight: 4)**
- ✓ **1.111.5 Maintain an effective data backup strategy (weight: 3)**
- ✓ **1.111.6 Maintain system time (weight: 4)**

Much of Linux system administration deals with handling mundane day-to-day tasks. Many of these tasks relate to users and groups: adding them, deleting them, configuring the environments, and so on. On a small system, you might perform such tasks infrequently, but on a busy system, you might adjust accounts frequently. In any event, you must know how to do these things. Another class of day-to-day tasks involves managing and reviewing *log files*. These are files that record details of system operations, such as remote logins. Log files can be invaluable debugging resources, but even if you aren't experiencing a problem, you should review them periodically to be sure everything's working as it should.

Many Linux tasks relate to time. Linux keeps time somewhat differently than some other OSs, and understanding how Linux treats time is important. So are the skills needed to set the time in Linux. (Some automated tools can be very helpful, but you must know how to configure them.) You can also tell Linux to run particular jobs at specific times in the future. This can be handy to help automate repetitive tasks, such as synchronizing data with other systems on a regular basis. You may even want to schedule backups on an automated basis. In fact, backups are very important, but they're often neglected.

Managing Users and Groups

Linux is a multi-user system, meaning that it relies on *accounts*—data structures and procedures used to identify individual users of a computer. Managing these accounts is a basic but important system administration skill. Before delving into the details, I describe a few basic concepts you should understand about user and group administration. With that out of the way, I describe the tools and configuration files that you employ to manage users and groups.

User and Group Concepts

Chances are you have a good basic understanding of accounts already. Fundamentally, Linux accounts are like accounts on Windows, Mac OS, and other OSs. Some websites use accounts, too. Nonetheless, a few details deserve explanation. These include Linux username conventions, the nature of Linux groups, and the way Linux maps the numbers it uses internally to the usernames and group names that people generally use.

Linux Usernames

Linux is fairly flexible about its usernames. Most versions of Linux support usernames consisting of any combination of upper- and lowercase letters, numbers, and many punctuation symbols, including periods and spaces. Some punctuation symbols, however, such as spaces, cause problems for certain Linux utilities, so it's generally best to avoid using punctuation in Linux usernames. Underscores (_) and periods (.) are relatively unlikely to cause problems and so are occasionally used. Also, usernames must begin with a letter, so a username such as `45u` is invalid, although `u45` is fine. Although usernames may consist of up to 32 characters, many utilities truncate usernames longer than 8 characters or so in their displays, so many administrators try to limit username length to 8 characters.

Linux treats usernames in a case-sensitive way. Therefore, a single computer can support both `ellen` and `Ellen` as separate users. This practice can lead to a great deal of confusion, however, so it's best to avoid creating accounts whose usernames differ only in case. In fact, the traditional practice is to use entirely lowercase letters in Linux usernames, such as `sally`, `sam`, `ellen`, and `george`. Usernames don't need to be based on first names, of course—you could use `sam_jones`, `s.jones`, `sjones`, `jones`, `jones17`, or `u238`, to name just a few possibilities. Most sites develop a standard method of creating usernames, such as using the first initial and the last name. Creating and following such a standard practice can help you locate an account that belongs to a particular individual. If your computer has many users, though, you may find a naming convention produces duplicates, particularly if your standard is to use initials to shorten usernames. You may therefore be forced to deviate from the standard or incorporate numbers to distinguish between all the Davids or Smiths of the world.

Groups: Linking Users Together for Productivity

Linux uses *groups* as a means of organizing users. In many ways, groups parallel users. In particular, they're defined in similar configuration files, have names similar to usernames, and are represented internally by numbers (as are accounts).

Groups are *not* accounts, however. Rather, groups are a means of organizing collections of accounts, largely as a security measure. Every file on a Linux system is associated with a specific group, and various permissions can be assigned to members of that group. For instance, group members (such as faculty at a university) might be allowed to read a file, but others (such as students) might be disallowed such access. Because Linux provides access to most hardware devices (such as serial ports and tape backup units) through files, this same mechanism can be used to control access to hardware.

Every group has anywhere from no members to as many members as there are users on the computer. Group membership is controlled through the `/etc/group` file. This file contains a list of groups and the members belonging to each group. The details of this file's contents are described in the section "Configuring Groups."

In addition to membership defined in `/etc/group`, each user has a default or primary group. The user's primary group is set in the user's configuration in `/etc/passwd` (the file that defines accounts). When users log onto the computer, their group membership is set to their primary groups. When users create files or launch programs, those files and running programs are associated with a single group—the current group membership. A user can still access files belonging

to other groups as long as the user belongs to that group and the group access permissions permit the access. To run programs or create files with other than the primary group membership, however, the user must run the `newgrp` command to switch current group membership. For instance, to change to the `project2` group, you might type the following:

```
$ newgrp project2
```

If the user typing this command is listed as a member of the `project2` group in `/etc/group`, the user's current group membership will change. Thereafter, files created by that user will be associated with the `project2` group. Alternatively, users can change the group associated with an existing file by using the `chgrp` or `chown` command, as described in Chapter 4.

This group structure enables you to design a security system that permits different collections of users to easily work on the same files while simultaneously keeping other users of the same computer from prying into files they should not be able to access. In a simple case, you might create groups for different projects, classes, or workgroups, with each user restricted to one of these groups. A user who needs access to multiple groups could be a member of each of these groups—for instance, a student who takes two classes could belong to the groups associated with each class, or a supervisor might belong to all the supervised groups.

Mapping UIDs and GIDs to Users and Groups

As mentioned earlier, Linux defines users and groups by numbers, referred to as *user IDs (UIDs)* and *group IDs (GIDs)*, respectively. Internally, Linux tracks users and groups by these numbers, not by name. For instance, the user `sam` might be tied to UID 523, and `ellen` might be UID 609. Similarly, the group `project1` might be GID 512, and `project2` might be GID 523. For the most part, these details take care of themselves—you use names, and Linux uses `/etc/passwd` or `/etc/group` to locate the number associated with the name. You may occasionally need to know how Linux assigns numbers when you tell it to do something, though. This is particularly true when you are troubleshooting or if you have cause to manually edit `/etc/passwd` or `/etc/group`.

Linux distributions reserve the first hundred user and group IDs (0–99) for system use. The most important of these is 0, which corresponds to `root` (both the user and the group). Subsequent low numbers are used by accounts and groups that are associated with specific Linux utilities and functions. For instance, UID 2 and GID 2 are generally the `daemon` account and group, respectively, which are used by various servers, and UID 8 and GID 12 are usually the `mail` account and group, which can be used by mail-related servers and utilities. Not all account and group numbers from 0 to 99 are in use; there are usually only one or two dozen accounts and a dozen or so groups used in this way. You can check your `/etc/passwd` and `/etc/group` files to determine which user and group IDs are so used.

> Aside from UID 0 and GID 0, UID and GID numbers aren't fully standardized. For instance, although UID 2 and GID 2 map to the daemon account and daemon group on Red Hat and SuSE, on Debian UID 2 and GID 2 map to the `bin` account and `bin` group; the daemon account and group correspond to UID 1 and GID 1. If you need to refer to a particular user or group, use the name rather than the number.

Beyond 100, user and group IDs are available for use by ordinary users and groups. Many distributions, however, reserve up to 500 or even 1000 for special purposes. Frequently, therefore, the first normal user account is assigned a UID of 500 or 1000. When you create additional accounts, the system typically locates the next-highest unused number, so the second user you create is UID 501, the third is 502, and so on. When you remove an account, that account's ID number may be reused, but the automatic account-creation tools typically don't do so if subsequent numbers are in use, leaving a gap in the sequence. This gap causes no harm unless you have so many users that you run out of ID numbers. (The limit is 65,536 users with the 2.2.x kernels and over 4.2 billion with the 2.4.x and later kernels, including root and other system accounts. The limit can be set lower in configuration files or because of limits in support programs.) In fact, reusing an ID number can cause problems if you don't clear away the old user's files—the new user will become the owner of the old user's files, which can lead to confusion.

Typically, GID 100 is users—the default group for some distributions. On any but a very small system with few users, you'll probably want to create your own groups. Because different distributions have different default ways of assigning users to groups, it's best that you familiarize yourself with your distribution's way of doing this and plan your own group-creation policies with this in mind. For instance, you might want to create your own groups within certain ranges of IDs to avoid conflicts with the distribution's default user- and group-creation processes.

It's possible to create multiple usernames that use the same UID or multiple group names that use the same GID. In some sense, these are different accounts or groups; they have different entries in /etc/passwd or /etc/group, so they can have different home directories, different passwords, and so on. Because these users or groups share IDs with other users or groups, though, they're treated identically in terms of file permissions. Unless you have a compelling reason to do so, you should avoid creating multiple users or groups that share an ID.

> **WARNING** Intruders sometimes create accounts with UID 0 to give themselves root privileges on the systems they invade. *Any* account with a UID of 0 is effectively the root account, with all the power of the superuser. If you spot a suspicious account in your /etc/passwd file with a UID of 0, your system has probably been compromised.

Configuring User Accounts

How frequently you'll do user maintenance depends on the nature of the system you administer. Some systems, such as small personal workstations, need changes very rarely. Others, such as large systems in environments in which users are constantly coming and going, may require daily maintenance. The latter situation would seem to require more knowledge of user account configuration tools, but even in a seldom-changing system, it's useful to know how to do these things so that you can do them quickly and correctly when you do need to add, modify, or delete user accounts.

> Some security-related account issues, such as the pwconv and related utilities, are covered in Chapter 7, "Documentation and Security."

Adding Users

Adding users can be accomplished through the useradd utility. (This program is called adduser on some distributions.) Its basic syntax is as follows:

```
useradd [-c comment] [-d home-dir] [-e expire-date] [-f inactive-days]
➥[-g initial-group] [-G group[,...]]  [-m [-k skeleton-dir] | -M]
➥[-p password] [-s shell]  [-u UID [-o]] [-r] [-n] username
```

> Some of these parameters modify settings that are valid only when the system uses shadow passwords. This is the standard configuration for most distributions today.

In its simplest form, you may type just **useradd *username***, where *username* is the username you want to create. The rest of the parameters are used to modify the default values for the system, which are stored in the file /etc/login.defs.

The parameters for the useradd command modify the program's operation in various ways:

Comment   The -c *comment* parameter passes the comment field for the user. Some administrators store public information like a user's office or telephone number in this field. Others store just the user's real name or no information at all.

Home directory   You specify the account's home directory with the -d *home-dir* parameter. This defaults to /home/*username* on most systems.

Account expiration date   Set the date on which the account will be disabled, expressed in the form *YYYY-MM-DD*, with the -e *expire-date* option. (Many systems will accept alternative forms, such as *MM-DD-YYYY*, as well.) The default is for an account that does not expire.

Inactive days   An account becomes completely disabled a certain number of days after a password expires. The -f *inactive-days* parameter sets the number of days. A value of -1 disables this feature and is the default.

Default group   You set the name or GID of the user's default group with the -g *default-group* option. The default for this value varies from one distribution to another.

Additional groups   The -G *group*[,...] parameter sets the names or GIDs of one or more groups to which the user belongs. These groups need not be the default group, and more than one may be specified by separating them with commas.

Home directory options   The system automatically creates the user's home directory if -m is specified. Normally, default configuration files are copied from /etc/skel, but you

may specify another template directory with the `-k` *skeleton-dir* option. Many distributions use `-m` as the default when running `useradd`.

Do not create a home directory   The `-M` option forces the system to *not* automatically create a home directory, even if `/etc/login.defs` specifies that this action is the default.

Encrypted password specification   The `-p` *encrypted-password* parameter passes the *pre-encrypted* password for the user to the system. The *encrypted-password* value will be added, *unchanged,* to the `/etc/passwd` or `/etc/shadow` file. This means that if you type an unencrypted password, it won't work as you probably expected. In practice, this parameter is most useful in scripts, which can encrypt a password (using `crypt`) and then send the encrypted result through `useradd`. The default value disables the account, so you must run `passwd` to change the user's password.

Default shell   Set the name of the user's default login shell with the `-s` *shell* option. On most systems, this defaults to `/bin/bash`.

Specify a UID   The `-u` *UID* parameter creates an account with the specified user ID value (*UID*). This value must be a positive integer, and it is normally above 500 for user accounts. System accounts typically have numbers below 100. The `-o` option allows the number to be reused so that two usernames are associated with a single UID.

System account creation   The `-r` parameter specifies the creation of a system account—an account with a value lower than UID_MIN, as defined in `/etc/login.defs`. (This is normally 100, 500, or 1000.) `useradd` also doesn't create a home directory for system accounts.

No user group   In some distributions, such as Red Hat, the system creates a group with the same name as the specified username. The `-n` parameter disables this behavior.

Suppose you've added a hard disk and mounted it as `/home2`. You want to create an account for a user named Sally in this directory and place her home directory on the new disk. You want to make the new user a member of the `project1` and `project4` groups, with default membership in `project4`. The user has also requested `tcsh` as her default shell. You might use the following commands to accomplish this goal:

```
# useradd -d /home2/sally -g project4 -G project1,project4  -s /bin/tcsh sally
# passwd sally
Changing password for user sally
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully
```

> **NOTE**   The passwd command asks for the password twice, but it does not echo what you type. This prevents somebody who sees your screen from reading the password off it. passwd is described in more detail in shortly, in "Setting a Password."

Modifying User Accounts

User accounts may be modified in many ways: You can directly edit critical files such as /etc/passwd, modify user-specific configuration files in the account's home directory, or use system utilities like those used to create accounts. You usually modify an existing user's account at the user's request or to implement some new policy or system change, such as moving home directories to a new hard disk. Sometimes, though, you must modify an account immediately after its creation in order to customize it in ways that aren't easily handled through the account-creation tools or because you realize you forgot a parameter to useradd.

Setting a Password

Although useradd provides the -p parameter to set a password, this tool is not very useful when directly adding a user because it requires a pre-encrypted password. Therefore, it's usually easiest to create an account in disabled form (by not using -p with useradd) and set the password after creating the account. This can be done with the passwd command, which has the following syntax:

```
passwd [-k] [-l] [-u [-f]] [-d] [-S] [username]
```

The parameters to this command enable you to modify its behavior:

Update expired accounts   The -k parameter indicates that the system should update an expired account.

Lock accounts   The -l parameter locks an account by prefixing the encrypted password with an exclamation mark (!). The result is that the user can no longer log into the account, but the files are still available and the change can be easily undone. This parameter is particularly handy if you want to temporarily suspend user access to an account—say, because you've spotted some suspicious activity involving the account or because you know a user won't be using the account for a while and you want to minimize the chance of it being abused in the interim.

Unlock accounts   The -u parameter unlocks an account by removing a leading exclamation mark. useradd creates accounts that are locked and have no password, so using this command on a fresh account would result in an account with no password. Normally, passwd doesn't allow this—it returns an error if you attempt it. Adding -f forces passwd to turn the account into one with no password.

Create accounts without passwords   The -d parameter removes the password from an account, rendering it password-less.

Display account information   The -S option displays information on the password for an account—whether or not it's set and what type of encryption it uses.

Ordinary users may use passwd to change their passwords, but many passwd parameters may only be used by root. Specifically, -l, -u, -f, -d, and -S are all off-limits to ordinary users. Similarly, only root may specify a username to passwd. When ordinary users run the program, they should omit their usernames and passwd will change the password for the user who ran the program. As a security measure, passwd asks for a user's old password before changing the password

when an ordinary user runs the program. This precaution is *not* taken when root runs the program so that the superuser may change a user's password without knowing the original password. This is necessary because the administrator normally doesn't know the user's password.

Linux passwords may consist of letters, numbers, and punctuation. Linux distinguishes between upper- and lowercase letters in passwords, which means you can use mixed-case passwords to improve security.

NOTE   Chapter 7 provides information on selecting good passwords.

EXERCISE 8.1

Creating User Accounts

This exercise explores the process of creating user accounts. After performing this exercise, you should be familiar with the text-mode Linux account-creation tools and be able to create new accounts, including preparing new users' home directories. To add and test a new account, follow these steps:

1. Log into the Linux system as a normal user.

2. Launch an xterm from the desktop environment's menu system, if you used a GUI login method.

3. Acquire root privileges. You can do this by typing **su** in an xterm, by selecting Session ➢ New Root Console from a Konsole, or by using sudo (if it's configured) to run the commands in the following steps.

4. Type **useradd -m** *username*, where *username* is the name you want to be associated with the account. This command creates an account. The -m parameter tells Linux to create a home directory for the user and fill it with default account configuration files.

5. Type **passwd** *username*. You'll be asked to enter a password for the user and to type it a second time. Enter a random string or select a password as described in "Setting a Password."

6. Press Ctrl+Alt+F2 to go to a fresh text-mode login screen. (If you're already using multiple virtual terminals, you may need to use a higher function key number than F2.)

7. Try logging in as the new user to verify that the account works properly.

In practice, creating accounts on a production system may require variations on this procedure. You may need to use additional options in step 4, for instance; consult the section "Adding Users" or the useradd man page for details on these options. Furthermore, setting the password may require changes. On a small system with few users, you may be able to create accounts in the presence of their future users, in which case the user can type the password in step 5. On other systems, you may need to generate passwords yourself and then give them to users in some way.

Using *usermod*

The usermod program closely parallels useradd in its features and parameters. This utility changes an existing account instead of creating a new one, though. The major differences between useradd and usermod are as follows:

- usermod allows the addition of a -m parameter when used with -d. The -d parameter alone changes the user's home directory, but it does not move any files. Adding -m causes usermod to move the user's files to the new location.

- usermod supports a -l parameter, which changes the user's login name to the specified value. For instance, typing **usermod sally -l sjones** changes the username from sally to sjones.

- You may lock or unlock a user's password with the -L and -U options, respectively. This duplicates functionality provided by passwd.

The usermod program changes the contents of /etc/passwd or /etc/shadow, depending on the option used. If -m is used, usermod also moves the user's files, as already noted.

> **WARNING**
>
> Changing an account's characteristics while the owner is logged in can have undesirable consequences. This is particularly true of the -d -m combination, which can cause the files a user is working on to move. Most other changes, such as changes to the account's default shell, simply don't take effect until the user has logged out and back in again.

If you change the account's UID, this action does *not* change the UIDs stored with a user's files. Because of this, the user may lose access to these files. You can manually update the UIDs on all files by using the chown command, as described in Chapter 4. Specifically, a command like the following, issued after changing the UID on the account sally, will restore proper ownership on the files in sally's home directory:

```
# chown -R sally /home/sally
```

This action does *not* change the ownership of files that aren't in sally's home directory. If you believe such files exist, you may need to track them down with the find command, as you'll see in the upcoming section "Deleting Accounts." Also, this command blindly changes ownership of *all* files in the /home/sally directory. This is probably desirable, but it's conceivable that some files in that directory *should* be owned by somebody else—say, because sally and another user are collaborating on a project.

When using the -G option to add a user to new groups, be aware that any groups *not* listed will be removed. The gpasswd command, described in the upcoming section "Using gpasswd," provides a way to add a user to one or more specific groups without affecting existing group memberships, and so it is generally preferable for this purpose.

Using *chage*

The chage command allows you to modify account settings relating to account expiration. It's possible to configure Linux accounts so that they automatically expire if either of two conditions is true:

- The password hasn't been changed in a specified period of time.
- The system date is past a predetermined time.

These settings are controlled through the chage utility, which has the following syntax:

```
chage [-l] [-m mindays] [-M maxdays] [-d lastday]  [-I inactivedays]
➥[-E expiredate] [-W warndays] username
```

The program's parameters modify the command's actions:

Display information   The -l option causes chage to display account expiration and password aging information for a particular user.

Set minimum time between password changes   The -m *mindays* parameter sets the minimum number of days between password changes. 0 indicates that a user can change a password multiple times in a day; 1 means that a user can change a password once a day; 2 means that a user may change a password once every two days; and so on.

Set maximum time between password changes   The -M *maxdays* parameter sets the maximum number of days that may pass between password changes. For instance, 30 would require a password change approximately once a month.

> **NOTE**   If the user changes a password before the deadline, the counter is reset from the password change date.

Set last password change date   The -d *lastday* parameter sets the last day a password was changed. This value is normally maintained automatically by Linux, but you can use this parameter to artificially alter the password change count. *lastday* is expressed in the format *YYYY/MM/DD* or as the number of days since January 1, 1970.

Maximum inactive days   The -I *inactivedays* parameter sets the number of days between password expiration and account disablement. An expired account may not be used or may force the user to change the password immediately upon logging in, depending on the distribution. A disabled account is completely disabled, however.

Set expiration date   You can set an absolute expiration date with the -E *expiredate* option. For instance, you might use **-E 2006/05/21** to have an account expire on May 21, 2006. The date may also be expressed as the number of days since January 1, 1970. A value of -1 represents no expiration date.

Set number of warning days   The -W *warndays* option sets the number of days before account expiration that the system will warn the user of the impending expiration. It's generally a good idea to use this feature to alert users of their situation, particularly if you make heavy use of password change expirations. Note that these warnings are usually only shown to text-mode login users; GUI login users, file share users, and so on usually don't see these messages.

The chage command can normally only be used by root. The one exception to this rule is if the -l option is used; this feature allows ordinary users to check their account expiration information.

Directly Modifying Account Configuration Files

User configuration files can be modified directly. The /etc/passwd and /etc/shadow files control most aspects of an account's basic features. Both files consist of a set of lines, one line per account. Each line begins with a username and continues with a set of fields, delimited by colons (:). Many of these items may be modified with usermod or passwd. A typical /etc/passwd entry resembles the following:

```
sally:x:529:100:Sally Jones:/home/sally:/bin/bash
```

Each field has a specific meaning, as follows:

Username The first field in each /etc/passwd line is the username (sally in this example).

Password The second field has traditionally been reserved for the password. Most Linux systems, however, use a shadow password system in which the password is stored in /etc/shadow. The x in the example's password field is an indication that shadow passwords are in use. In a system that does not use shadow passwords, an encrypted password will appear here instead.

UID Following the password is the account's user ID (529 in this example).

Primary GID The default login group ID is next in the /etc/passwd line for an account. The example uses a primary GID of 100.

Comment The comment field may have different contents on different systems. In the preceding example, it's the user's full name. Some systems place additional information here, in a comma-separated list. Such information might include the user's telephone number, office number, title, and so on.

Home directory The user's home directory is next up in the list.

Default shell The default shell is the final item on each line in /etc/passwd. This is normally /bin/bash, /bin/tcsh, or some other common command shell. It's possible to use something unusual here, though. For instance, many systems include a shutdown account with /bin/shutdown as the shell. If you log into this account, the computer immediately shuts down. You can create user accounts with a shell of /bin/false, which prevents users from logging in as ordinary users but leaves other utilities intact. Users can still receive mail and retrieve it via a remote mail retrieval protocol like POP or IMAP, for instance. A variant on this scheme uses /bin/passwd so that users may change their passwords remotely but cannot actually log in using a command shell.

You can directly modify any of these fields, although in a shadow password system, you probably do *not* want to modify the password field; you should make password-related changes via passwd so that they can be properly encrypted and stored in /etc/shadow. As with changes initiated via usermod, it's best to change /etc/passwd directly only when the user in question isn't logged in, to prevent a change from disrupting an ongoing session.

Like /etc/passwd, /etc/shadow may be edited directly. An /etc/shadow line resembles the following:

```
sally:E/moFkeT5UnTQ:12269:0:-1:7:-1:-1:
```

Most of these fields correspond to options set with the `chage` utility, although some are set with `passwd`, `useradd`, or `usermod`. The meaning of each colon-delimited field of this line is as follows:

Username   Each line begins with the username. Note that the UID is *not* used in `/etc/shadow`; the username links entries in this file to those in `/etc/passwd`.

Password   The password is stored in encrypted form, so it bears no obvious resemblance to the actual password. An asterisk (`*`) or exclamation mark (`!`) denotes an account with no password (that is, the account doesn't accept logins—it's locked). This is common for accounts used by the system itself.

> If you've forgotten the `root` password for a system, you can boot with an emergency recovery system and copy the contents of a password field for an account whose password you do remember. You can then boot normally, log in as `root`, and change the password. In a real pinch, you can delete the contents of the password field, which results in a `root` account with *no* password (that is, none is required to log in). Be *sure* to *immediately* change the `root` password after rebooting if you do this, though!

Last password change   The next field (`12269` in this example) is the date of the last password change. This date is stored as the number of days since January 1, 1970.

Days until change allowed   The next field (`0` in this example) is the number of days before a password change is allowed.

Days before change required   This field is the number of days after the last password change before another password change is required.

Days warning before password expiration   If your system is configured to expire passwords, you may set it to warn the user when an expiration date is approaching. A value of `7`, as in the preceding example, is typical.

Days between expiration and deactivation   Linux allows for a gap between the expiration of an account and its complete deactivation. An expired account either cannot be used or requires that the user change the password immediately after logging in. In either case, its password remains intact. A deactivated account's password is erased, and the account cannot be used until it's reactivated by the system administrator.

Expiration date   This field shows the date on which the account will be expired. As with the last password change date, the date is expressed as the number of days since January 1, 1970.

Special flag   This field is reserved for future use and is normally not used or contains a meaningless value. This field is empty in the preceding example.

For fields relating to day counts, values of `-1` or `99999` typically indicate that the relevant feature has been disabled. The `/etc/shadow` values are generally best left to modification through the `usermod` or `chage` commands because they can be tricky to set manually—for instance, it's easy to forget a leap year or the like when computing a date as the number of days since January 1, 1970. Similarly, because of its encrypted nature, the password field

cannot be edited effectively except through `passwd` or similar utilities. (You could cut and paste a value from a compatible file or use `crypt` yourself, but it's generally easier to use `passwd`. Copying encrypted passwords from other systems is also somewhat risky because it means that the users will have the same passwords on both systems, and this fact will be obvious to anybody who's acquired both encrypted password lists.)

> **WARNING** The `/etc/shadow` file is normally stored with very restrictive permissions, such as `rw-------`, with ownership by `root`. This fact is critical to the shadow password system's utility since it keeps non-`root` users from reading the file and obtaining the password list, even in an encrypted form. Therefore, if you manually modify `/etc/shadow`, be sure it has the correct permissions when you're done.

Real World Scenario

Network Account Databases

Many networks employ network account databases. Such systems include the Network Information System (NIS), an update to this system called NIS+, the Lightweight Directory Access Protocol (LDAP), Kerberos realms, Windows NT 4.0 domains, and Active Directory (AD) domains. All of these systems move account database management onto a single centralized computer (often with one or more backup systems). The advantage of this approach to account maintenance is that users and administrators need not deal with maintaining accounts independently on multiple computers. A single account database can handle accounts on dozens (or even hundreds or thousands) of different computers, greatly simplifying day-to-day administrative tasks and simplifying users' lives. Using such a system, though, means that most user accounts won't appear in `/etc/passwd` and `/etc/shadow` and groups may not appear in `/etc/group`. (These files will still hold information on local system accounts and groups, though.)

Linux can participate in these systems. In fact, some distributions provide options to enable such support at OS installation time. Typically, you must know the name or IP address of the server that hosts the network account database, and you must know what protocol that system uses. You may also need a password or some other protocol-specific information, and the server may need to be configured to accept accesses from the Linux system you're configuring.

Activating use of such network account databases after installing Linux is a complex topic. It involves installing appropriate software, modifying the `/etc/nsswitch.conf` file, and modifying the Pluggable Authentication Module (PAM) configuration files in `/etc/pam.d`. Such systems often alter the behavior of tools such as `passwd` and `usermod` in subtle or not-so-subtle ways. If you need to use such a system, you'll have to consult documentation specific to the service you intend to use. My book *Linux in a Windows World* (2005) covers this topic for Windows NT 4.0 domains, LDAP, and Kerberos, and Mark Minasi and Dan York's *Linux for Windows Administrators* (Sybex, 2002) covers this topic for Windows NT 4.0 domains and NIS.

Deleting Accounts

On the surface, deleting user accounts is easy. You may use the `userdel` command to do the job of removing a user's entries from `/etc/passwd` and, if the system uses shadow passwords, `/etc/shadow`. The `userdel` command takes just one parameter: `-r`. This parameter causes the system to remove all files from the user's home directory, as well as the home directory itself. Thus, removing a user account such as `sally` is easily accomplished with the following command:

```
# userdel -r sally
```

You may omit the `-r` parameter if you want to preserve the user's files. There is one potential complication, however: Users may create files *outside* their home directories. For instance, many programs use the `/tmp` directory as "scratch space," so user files often wind up there. These files will be deleted automatically after a certain period, but you may have other directories in which users might store files. To locate all such files, you can use the `find` command with its `-uid` parameter. For instance, if `sally` had been UID 529, you might use the following command to locate all her files:

```
# find / -uid 529
```

The result will be a list of files owned by UID 529 (formerly `sally`). You can then go through this list and decide what to do with the files—change their ownership to somebody else, delete them, back them up to floppy, or what have you. It's wise to do *something* with these files, though, or else they may be assigned ownership to another user if Sally's UID is reused. This could become awkward if the files exceed the new user's disk quota or if they contain information that the new user should not have—such a person might mistakenly be accused of indiscretions or even crimes.

A few servers—most notably Samba—may keep their own list of users. If you run such a server, it's best to remove the user's entry from that server's user list when you remove the user's main account. In the case of Samba, this is normally done by manually editing the `smbpasswd` file (usually located in `/etc`, `/etc/samba`, or `/etc/samba.d`) and deleting the line corresponding to the user in question or using the `smbpasswd` command and its `-x` option, as in **smbpasswd -x sally** to delete the `sally` account from Samba's database.

Configuring Groups

Linux provides group configuration tools that parallel those for user accounts in many ways. Groups are not accounts, however, so many features of these tools differ. Likewise, you can create or modify groups by directly editing the configuration files in question. Their layout is similar to that for account control files, but the details differ.

Adding Groups

Linux provides the `groupadd` command to add a new group. This utility is similar to `useradd` but has fewer options. The `groupadd` syntax is shown here:

```
groupadd [-g GID [-o]] [-r] [-f] groupname
```

The parameters to this command enable you to adjust its operation:

Specify a GID   You can provide a specific GID with the -g *GID* parameter. If you omit this parameter, `groupadd` uses the next available GID. Normally, the GID you specify must be unused by other groups, but the -o parameter overrides this behavior, allowing you to create multiple groups that share one GID.

Create a sub-500 GID   The -r parameter instructs `groupadd` to create a group with a GID of less than 500. Not all distributions support this option; it was added by Red Hat and has been used on some related distributions. Red Hat uses GIDs of 500 and above for user private groups (that is, groups named after individual users), hence the -r parameter.

Force creation   Normally, if you try to create a group that already exists, `groupadd` returns an error message. The -f parameter suppresses that error message. Not all versions of `groupadd` support this parameter.

In most cases, you'll create groups without specifying any parameters except for the group name itself:

```
# groupadd project3
```

This command creates the `project3` group, giving it whatever GID the system finds convenient—usually the highest existing GID plus 1. Once you've done this, you can add users to the group, as described in the next section. When you add new users, you can add them directly to the new group with the -g and -G parameters to `useradd`, described earlier.

Modifying Group Information

Group information, like user account information, may be modified either using utility programs or by directly editing the underlying configuration file, /etc/group. As with creation, there are fewer options for modifying groups than for modifying accounts, and the utilities and configuration files are similar. In fact, `usermod` is one of the tools that's used to modify groups.

Using *groupmod* and *usermod*

The `groupmod` command modifies an existing group's settings. Its syntax is shown here:

```
groupmod [-g GID [-o]] [-n newgroupname] oldgroupname
```

The options to this command modify its operation:

Specify a GID   Specify the new group ID using the -g *GID* option. `groupmod` returns an error if you specify a new group ID that's already in use, unless you include the -o parameter, in which case you can create two groups that share a single GID.

Specify a group name   Specify a new group name with the -n *newgroupname* option.

One of the most common group manipulations you'll perform, however, is not handled through `groupmod`; it's done with `usermod`. Specifically, `usermod` allows you to add a user to a group with its -G parameter. For instance, the following command sets `sally` to be a member of the `users`, `project1`, and `project4` groups, and it removes her from all other groups:

```
# usermod -G users,project1,project4 sally
```

> **WARNING** Be sure to list all the user's current groups in addition to any groups to which you want to add the user. Omitting any of the user's current groups will remove the user from those groups. You can discover the groups to which a user currently belongs with the groups command, as in **groups sally**. To avoid accidentally omitting a group, many system administrators prefer to modify the /etc/group file in a text editor or use gpasswd. Both options allow you to add users to groups without specifying a user's existing group memberships.

Using *gpasswd*

The gpasswd command is the group equivalent to passwd. The gpasswd command also enables you to modify other group features and to assign *group administrators*—users who may perform some group-related administrative functions for their groups. The basic syntax for this command is:

```
gpasswd [-a user] [-d user] [-R] [-r] [-A user[,...]] [-M user[,...]]
➥group
```

The options for this command modify its actions:

Add a user   The -a *user* option adds the specified user to the specified group.

Delete a user   The -d *user* option deletes the specified user from the specified group.

Disallow newgrp additions   The -R option configures the group to not allow anybody to become a member through newgrp.

Remove password   The -r option removes the password from a group.

Add group administrators   The root user may use the -A *user*[,...] parameter to specify group administrators. Group administrators may add members to and remove members from a group and change the group password. Using this parameter completely overwrites the list of administrators, so if you want to add an administrator to an existing set of group administrators, you must specify *all* of their usernames.

Add users   The -M *user*[,...] option works like -A, but it also adds the specified user(s) to the list of group members.

If entered without any parameters except a group name, gpasswd changes the password for the group. Group passwords enable you to control temporary membership in a group, as granted by newgrp. Ordinarily, members of a group may use newgrp to change their current group membership (affecting the group of files they create). If a password is set, even those who aren't members of a group may become temporary group members; newgrp prompts for a password that, if entered correctly, gives the user temporary group membership.

Unfortunately, some of these features are not implemented correctly in all distributions. In particular, password entry by non-group members sometimes does *not* give group membership—the system responds with an access denied error message. The -R option also sometimes doesn't work correctly—group members whose primary group membership is with another group may still use newgrp to set their primary group membership.

Directly Modifying Group Configuration Files

Group information is stored primarily in the /etc/group file. Like account configuration files, the /etc/group file is organized as a set of lines, one line per group. A typical line in this file resembles the following:

```
project1:x:501:sally,sam,ellen,george
```

Each field is separated from the others by a colon. The meanings of the four fields are as follows:

Group name   The first field (project1 in the preceding example) is the name of the group.

Password   The second field (x in the preceding example) is the group password. Distributions that use shadow passwords typically place an x in this field; others place the encrypted password directly in this field.

GID   The group ID number (in this example's case, 501) goes in this field.

User list   The final field is a comma-separated list of group members.

Users may also be members of a group based on their own /etc/passwd file primary group specification. For instance, if user george had project1 listed as his primary group, he need not be listed in the project1 line in /etc/group. If user george uses newgrp to change to another group, though, he won't be able to change back to project1 unless he's listed in the project1 line in /etc/group.

Systems with shadow passwords also use another file, /etc/gshadow, to store shadow password information on groups. This file stores the shadow password and information on group administrators, as described earlier in "Using gpasswd."

> If you configure Linux to use a network account database, the /etc/group file will be present and may define groups important for the system's basic operation. As with /etc/passwd and /etc/shadow, though, important user groups are likely to be defined only on the network account server, not in /etc/group.

Deleting Groups

Deleting groups is done via the groupdel command, which takes a single parameter: a group name. For instance, **groupdel project3** removes the project3 group. You can also delete a group by editing the /etc/group file (and /etc/gshadow, if present) and removing the relevant line for the group. It's generally better to use groupdel, though, because groupdel checks to see if the group is any user's primary group. If it is, groupdel refuses to remove the group; you must change the user's primary group or delete the user account first.

As with deleting users, deleting groups can leave "orphaned" files on the computer. You can locate them with the find command, which is described in more detail in Chapter 2. For instance, if the deleted group had used a GID of 503, you can find all the files on the computer with that GID by using the following command:

```
# find / -gid 503
```

Once you've found any files with the deleted group's ownership, you must decide what to do with them. In some cases, leaving them alone won't cause any immediate problems, but if the GID is ever reused, it could lead to confusion and even security breaches. Therefore, it's usually best to delete the files or assign them other group ownership using the `chown` or `chgrp` command.

Tuning User and System Environments

Chapter 6 described `bash` configuration files, such as `/etc/profile`, `/etc/bashrc`, `~/.profile`, and `~/.bashrc`. These files control the various `bash` options, including environment variables—named variables that hold data for the benefit of many programs. For instance, you might set the `EDITOR` environment variable to the name of your favorite text editor. Some (but not all) programs that launch editors will pay attention to this environment variable and launch the editor you specify.

As a system administrator, you can change the system-wide `bash` configuration files to add, remove, or change the environment variables that all users receive. Generally speaking, you should do so because the documentation for a specific programs indicates that it uses particular environment variables; however, you might want to peruse Chapter 6's Table 6.2 to see what common environment variables you might want to adjust. You can also see all your current environment variables by typing **env**. (The list is rather long, so you may want to pipe it through `less`, as in **env | less**.)

> Various programs set environment variables themselves, and some are maintained automatically by bash. For instance, bash maintains the PWD environment variable, so you shouldn't try to set it in a configuration script. Also, be aware that adjusting the bash configuration files only affects bash. If a user's default shell is something else, or if a user doesn't use a text-mode shell (say, if the user logs into X and launches programs from a GUI menu), setting environment variables in the bash configuration files will do no good.

In addition to setting default environment variables and otherwise modifying users' text-mode login environment by adjusting their `bash` configuration files, you can adjust the default set of files created by `useradd`. As described earlier, in "Adding Users," `useradd` copies files from the *skeleton directory* (`/etc/skel` by default) into a newly created home directory. Typically, `/etc/skel` contains a handful of user configuration files, such as `.bashrc`. You can add any files (and even directories) to this directory that you like, including user configuration files, a starting directory tree, a `README` file for new users, and anything else you like. Because these files are copied into users' own home directories and they're given ownership of the copies, users will be able to read, change, and even delete their copies of these files. Thus, you shouldn't place any options in these files that are sensitive from a security point of view or that users should not be able to change. (In truth, entries you place in global `bash` configuration files can easily be overridden by individual users, as well.) Also, be aware that any changes you make to the global files won't automatically be moved into existing users' copies of these files; changes will affect only the files received by new users. This fact makes the global files (such as `/etc/profile`) preferable to `/etc/skel` for any changes to system defaults you want to implement systemwide, particularly if you expect you'll ever want to modify your changes.

Using System Log Files

Linux maintains log files that record various key details about system operation. You may be able to begin using log files immediately, but knowing how to change the log file configuration can also be important. You do this by configuring the `syslogd` *daemon* (a daemon is a program that runs continuously in the background waiting for some event to trigger it to perform some action). Some servers and other programs perform their own logging and so must be configured independently of `syslogd`. You may even want to configure one computer to send its log files to another system as a security measure. You should also be aware of issues surrounding log file rotation; if your computer doesn't properly manage existing log files, they can grow to consume all your available disk space, at least on the partition on which they're stored. In addition to configuring logging, you must be able to use the log files that the system generates.

Understanding *syslogd*

Most Linux systems employ a special daemon to handle log maintenance in a unified way. The traditional Linux system logger is `syslogd`, which is often installed from a package called `sysklogd`. The `syslogd` daemon handles messages from servers and other user-mode programs. It's usually paired with a daemon called `klogd`, which is usually installed from the same `sysklogd` package as `syslogd`. The `klogd` daemon manages logging of kernel messages.

> **NOTE** Other choices for system loggers exist. For instance, `syslog-ng` is a replacement that supports advanced filtering options, and `metalog` is another option. This chapter describes the traditional `syslogd` logger. Others are similar in principle, and even in some specific features, but differ in many details.

The basic idea behind a system logger is to provide a unified means of handling log files. The daemon runs in the background and accepts data delivered from servers and other programs that are configured to use the log daemon. The daemon can then use information provided by the server to classify the message and direct it to an appropriate log file. This configuration enables you to consolidate messages from various servers in a handful of standard log files, which can be much easier to use and manage than potentially dozens of log files from the various servers running on the system.

In order to work, of course, the log daemon must be configured. In the case of `syslogd`, this is done through the `/etc/syslog.conf` file. The next section describes this file's format in more detail.

Setting Logging Options

The format of the `/etc/syslog.conf` file is conceptually simple but provides a great deal of power. Comment lines, as in many Linux configuration files, are denoted by a hash mark (#). Noncomment lines take the following form:

```
facility.priority    action
```

In this line, the *facility* is a code word for the type of program or tool that has generated the message to be logged; the *priority* is a code word for the importance of this message; and the *action* is a file, remote computer, or other location that's to accept the message. The *facility* and *priority* are often referred to collectively as the selector.

Valid codes for the *facility* are `auth`, `authpriv`, `cron`, `daemon`, `kern`, `lpr`, `mail`, `mark`, `news`, `security`, `syslog`, `user`, `uucp`, and `local0` through `local7`. Many of these names refer to specific servers or program classes. For instance, mail servers and other mail-processing tools typically log using the `mail` facility. Most servers that aren't covered by more specific codes use the `daemon` facility. The `security` facility is identical to `auth`, but `auth` is the preferred name. The `mark` facility is reserved for internal use. An asterisk (*) refers to all facilities. You can specify multiple facilities in one selector by separating the facilities with commas (`,`).

Valid codes for the *priority* are `debug`, `info`, `notice`, `warning`, `warn`, `error`, `err`, `crit`, `alert`, `emerg`, and `panic`. The `warning` priority is identical to `warn`, `error` is identical to `err`, and `emerg` is identical to `panic`. The `error`, `warn`, and `panic` priority names are deprecated; you should use their equivalents instead. Other than these identical pairs, these priorities represent ascending levels of importance. The `debug` level logs the most information; it's intended, as the name implies, for debugging programs that are misbehaving. The `emerg` priority logs the most important messages, which indicate very serious problems. When a program sends a message to the system logger, it includes a priority code; the logger logs the message to a file if you've configured it to log messages of that level or higher. Thus, if you specify a *priority* code of `alert`, the system will log messages that are classified as `alert` or `emerg` but not messages of `crit` or below. An exception to this rule is if you precede the priority code by an equal sign (`=`), as in `=crit`, which describes what to do with messages of `crit` priority *only*. An exclamation mark (`!`) reverses the meaning of a match. For instance, `!crit` causes messages *below* `crit` priority to be logged. A *priority* of * refers to all priorities.

You can specify multiple selectors for a single action by separating the selectors by a semicolon (`;`). Note that commas are used to separate multiple facilities within a single selector, whereas semicolons are used to separate multiple selectors as a whole. Examples of complete selectors appear shortly.

Most commonly, the *action* is a filename, typically in the `/var/log` directory tree. Other possibilities include a device filename for a console (such as `/dev/console`) to display data on the screen, a remote machine name preceded by an at sign (`@`), and a list of usernames of individuals who should see the message if they're logged in. For the last of these options, an asterisk (*) means all logged-in users.

Some examples should help clarify these rules. First is a fairly ordinary and simple entry:

```
mail.*          /var/log/mail
```

This line sends all log entries identified by the originating program as related to mail to the `/var/log/mail` file. Most of the entries in a default `/etc/syslog.conf` file resemble this one. Together, they typically cover all of the facilities mentioned earlier. Some messages may be handled by multiple rules. For instance, another rule might look like this one:

```
*.emerg         *
```

This line sends all `emerg`-level messages to the consoles of all users who are logged into the computer using text-mode tools. If this line and the earlier `mail.*` selector are both present, `emerg`-level messages related to mail will be logged to `/var/log/mail` *and* displayed on users' consoles.

A more complex example logs kernel messages in various ways, depending on their priorities:

```
kern.*                /var/log/kernel
kern.crit             @logger.pangaea.edu
kern.crit             /dev/console
kern.info;kern.!err   /var/log/kernel-info
```

The first of these rules logs all kernel messages to `/var/log/kernel`. The next two lines relate to high-priority (`crit` or higher) messages from the kernel. The first of these lines sends such messages to `logger.pangaea.edu`. (This system must be configured to accept remote logs, which is a topic not covered in this book.) The second of these lines sends a copy of these messages to `/dev/console`, which causes them to be displayed on the computer's main text-mode console display. Finally, the last line sends messages that are between `info` and `err` in priority to `/var/log/kernel-info`. Because `err` is the priority immediately above `crit`, and because `info` is the lowest priority, these four lines cause all kernel messages to be logged two or three times: once to `/var/log/kernel` as well as to either the remote system and the console *or* to `/var/log/kernel-info`.

Most distributions ship with reasonable system logger settings, but you may want to examine these settings, and perhaps adjust them. If you change them, though, be aware that you may need to change some other tools. For instance, all major distributions ship with tools that help rotate log files. If you change the files to which `syslogd` logs messages, you may need to change your log file rotation scripts as well. This topic is covered in the next section.

In addition to the system logger's options, you may be able to set logging options in individual programs. For instance, you might tell programs to record more or less information or to log routine information at varying priorities. Some programs also provide the means to log via the system log daemon or via their own mechanisms. Details vary greatly from one program to another, so you should consult the program's documentation for details.

> Most programs that use the system log daemons are servers and other system tools. Programs that individuals run locally seldom log data via the system log daemon, although there are some exceptions to this rule, such as the Fetchmail program for retrieving e-mail from remote servers.

Rotating Log Files

Log files are intended to retain information on system activities for a reasonable period of time; however, system logging daemons provide no means to control the size of log files. Left unchecked, log files can therefore grow to consume all the available space on the partition on which they reside. To avoid this problem, Linux systems employ *log file rotation* tools. These tools rename and optionally compress the current log files, delete old log files, and force the logging system to begin using new log files.

    The most common log rotation tool is a package called `logrotate`. This program is typically called on a regular basis via a `cron` job. (The upcoming section "Running Jobs in the Future" describes `cron` jobs in more detail.) The `logrotate` program consults a configuration file called `/etc/logrotate.conf`, which includes several default settings and typically refers to files in `/etc/logrotate.d` to handle specific log files. A typical `/etc/logrotate.conf` file includes several comment lines, denoted by hash marks (#), as well as lines to set various options, as illustrated by Listing 8.1.

Listing 8.1: Sample */etc/logrotate.conf* File

```
# Rotate logs weekly
weekly

# Keep 4 weeks of old logs
rotate 4

# Create new log files after rotation
create

# Compress old log files
compress

# Refer to files for individual packages
include /etc/logrotate.d

# Set miscellaneous options
notifempty
nomail
noolddir

# Rotate wtmp, which isn't handled by a specific program
/var/log/wtmp {
   monthly
   create 0664 root utmp
   rotate 1
}
```

    Most of the lines in Listing 8.1 set options that are fairly self-explanatory or that are well explained by the comments that immediately precede them—for instance, the `weekly` line sets the default log rotation interval to once a week. If you see an option in your file that you don't understand, consult the `man` page for `logrotate`.

NOTE   Because log file rotation is handled by `cron` jobs that typically run late at night, it won't happen if a computer is routinely turned off at the end of the day. This practice is common with Windows workstations but is uncommon with servers. Either Linux workstations should be left running overnight as a general practice or some explicit steps should be taken to ensure that log rotation occurs despite routine shutdowns. The anacron utility, described in the upcoming section "Using anacron," is particularly well suited to this task.

The last few lines of Listing 8.1 demonstrate the format for the definition for a specific log file. These definitions begin with the filename for the file (multiple filenames may be listed, separated by spaces), followed by an open curly brace ({). They end in a close curly brace (}). Intervening lines set options that may override the defaults. For instance, the `/var/log/wtmp` definition in Listing 8.1 sets the `monthly` option, which tells the system to rotate this log file once a month, overriding the default `weekly` option. Such definitions are common in the individual files in `/etc/logrotate.d`, which are typically owned by the packages whose log files they rotate. Examples of features that are often set in these definitions include:

Rotated file naming   Ordinarily, rotated log files acquire numbers, such as `messages.1` for the first rotation of the `messages` log file. Using the `dateext` option causes the rotated log file to obtain a date code instead, as in `messages-20051005` for the rotation performed on October 5, 2005.

Compression options   As already noted, `compress` causes `logrotate` to compress log files to save space. This is done using `gzip` by default, but you can specify another program with the `compresscmd` keyword, as in `compresscmd bzip2` to use `bzip2`. The `compressoptions` keyword enables you to pass options to the compression command (say, to improve the compression ratio).

Creating new log files   The `create` option causes `logrotate` to create a new log file for use by the system logger or program. This option takes a file mode, owner, and group as additional options. Some programs don't work well with this option, though. Most of them use the `copytruncate` option instead, which tells `logrotate` to copy the old log file to a new name and then clear all the data out of the original file.

Time options   The `daily`, `weekly`, and `monthly` options tell the system to rotate the log files at the specified intervals. These options aren't always used, though; some configurations use a size threshold rather than a time threshold for when to rotate log files.

Size options   The `size` keyword sets a maximum size for a log file. It takes a size in bytes as an argument (adding k or M to the size changes it to kilobytes or megabytes, respectively). For instance, `size 100k` causes `logrotate` to rotate the file when it reaches 100KB in size.

Rotation options   The `rotate x` option causes *x* copies of old log files to be maintained. For instance, if you set `rotate 2` for the `/var/log/messages` file, `logrotate` will maintain `/var/log/messages.1` and `/var/log/messages.2` in addition to the active `/var/log/messages` file. When that file is rotated, `/var/log/messages.2` is deleted, `/var/log/messages.1` is

renamed to /var/log/messages.2, /var/log/messages becomes /var/log/messages.1, and a new /var/log/messages is created.

Mail options   If you use mail *address*, logrotate will e-mail a log file to the specified address when it's rotated out of existence. Using nomail causes the system to not send any e-mail; the log is quietly deleted.

Scripts   The prerotate and postrotate keywords both begin a series of lines that are treated as scripts to be run immediately before or after log file rotation, respectively. In both cases, these scripts end with the endscript keyword. These commands are frequently used to force syslogd or a server to begin using a new log file.

In most cases, servers and other programs that log data either do so via the system logging daemon or ship with a configuration file that goes in /etc/logrotate.d to handle the server's log files. These files usually do a reasonable job; however, you might want to double-check them. For instance, you might discover that your system is configured to keep too many or too few old log files for your taste, in which case adjusting the rotate option is in order. You should also check the /var/log directory and its subdirectories every now and then. If you see huge numbers of files accumulating, or if files are growing to unacceptable size, you may want to check the corresponding logrotate configuration files. If an appropriate file doesn't exist, create one. Use a working file as a template, modifying it for the new file. Pay particular attention to the prerotate or postrotate scripts; you may need to consult the documentation for the program that's creating the log file to learn how to force that program to begin using a new log file.

In most cases, log files remain on the computer that recorded them. Sometimes, though, you may want to copy such files off-site. The easiest way to do this may be to reconfigure the log daemon to send the messages you want to archive to another system, as described in "Setting Logging Options." Another possibility is to create a cron job (as described later, in "foo") to copy files to another system using a network share, ssh, or some other network tool. You can also manually copy log files onto removable disks, if you like. There are few technical reasons to archive log files for more than a few weeks—only if a problem escapes your notice for a long time will they be useful. Managers or lawyers might want to keep them around for the long term for business or legal reasons, though.

Reviewing Log File Contents

Log files do no good if they simply accumulate on the system. Their purpose is to be used as a means of identifying problems. When a server isn't responding as you expect, when a computer refuses logins it should be accepting (or accepting logins it should be refusing), or when a system's network interface isn't coming up (to name just three types of problems), you should check your log files as part of your troubleshooting procedures. Several procedures, many of which involve tools described elsewhere in this book, can help you do so:

Paging through whole log files   You can use a pager program, such as less (described in Chapter 1), to view the entire contents of a log file. A text editor can fill the same role.

Searching for keywords   You can use `grep` (described in Chapter 1) to pull lines that contain keywords out of log files. This can be particularly handy when you don't know which log file is likely to hold an entry. For instance, typing **`grep eth0 /var/log/*`** locates all lines in all files in the /var/log directory that contain the string `eth0`.

Examining the start or end of a file   You can use the `head` or `tail` command (described in Chapter 1) to examine the first or last several lines of a log file. The `tail` command is particularly handy; you can use it to look at the last few entries just after you take some action that you expect to produce some diagnostic log file entries.

Monitoring log files   In addition to checking the last few lines of a log file, `tail` can monitor a file on an ongoing basis, echoing lines to the screen as they're added to the file. You do this with the `-f` option to `tail`, as in **`tail -f /var/log/messages`**.

Advanced log analysis tools   Various packages exist expressly for the purpose of analyzing log files. For instance, there's Logcheck, which is part of the Sentry Tools package (`http://sourceforge.net/projects/sentrytools/`). This package comes with some distributions, such as Mandriva and Debian. Unfortunately, it requires a fair amount of customization for your own system, so it's most easily implemented if it comes with your distribution, preconfigured for its log file format.

Log file analysis is a skill that's best learned through experience. Many log file messages are cryptic, and they can be cryptic in different ways for different programs. For instance, consider these entries:

```
Apr 14 23:17:00 speaker /USR/SBIN/CRON[6026]: (george) CMD
➥ (/usr/bin/fetchmail -f /home/george/.fetchmailrc > /dev/null)
Apr 14 23:17:52 speaker sshd[6031]: Accepted publickey for george from
➥ ::ffff:192.168.1.3 port 48139 ssh2
```

These two lines relate to two entirely different events, but they have a similar format. Both entries begin with a time stamp and the name of the computer on which the activity occurred (`speaker` in this example). Next on each line is an identifier for the program that logged the activity, including its process ID (PID) number: /USR/SBIN/CRON[6026] and `sshd[6031]` in this example. Note that these names are generated by the programs that create the activity, so they aren't necessarily consistent or even fully accurate. For instance, there is no /USR/SBIN/CRON program, although there *is* a /usr/sbin/cron program. (Recall that Linux has a case-sensitive filesystem.)

All of this information helps you identify what program logged the entry and when it did so. The rest of the log entry contains the actual logged data. The first entry in this example is from the `cron` utility, and it identifies a program run on behalf of `george`—specifically, `cron` ran the `fetchmail` program, passed it the name of a configuration file via the `-f` option, and redirected the output to /dev/null. The second entry (for `sshd`) identifies a login from 192.168.1.3 on port 38139, again involving the user `george`.

You can use entries like these to help identify malfunctioning servers, spot security breaches, and otherwise debug your system. Doing so, though, requires at least some familiarity with the normal log file contents, as well as other system details. For instance, in the preceding example,

if your system has no george account, these entries should both be suspicious; however, you must be familiar enough with the format of the entries to spot that george is a username (or be able to work it out). You must also know that your system should have no george account.

Overall, you should probably examine your log files from time to time to become familiar with their contents. This will help you spot abnormalities when the system begins misbehaving or when you want to use log files to help track down an unwelcome visitor.

> Log file entries can be conspicuous by their absence as well as by suspicious content within them. Intruders often try to cover their tracks by editing log files to remove the entries that betray their unauthorized accesses. Sometimes, though, they're sloppy about this and they just delete all the log entries from the time in question. If you notice unusual gaps in your log files, such as a gap of an hour with no entries on a system that normally logs a couple dozen entries in that period, you may want to investigate further.

Maintaining the System Time

Linux depends on its system clock more than many OSs. Tools such as cron and at (described later, in "Running Jobs in the Future") run programs at specified times, the make development tool uses files' time stamps to determine which ones need attention, and so on. Thus, you should be familiar with how Linux deals with time, how to set the time zone, how to set the time, and how to keep the clock accurate.

Linux Time Concepts

The *x*86 computers that most often run Linux, as well as most other computers of this general class, have two built-in clocks. The first of these clocks, sometimes called the *hardware clock,* maintains the time while the computer is turned off. When you boot Linux, it reads the hardware clock and sets the *software clock* to the value it retrieves. The software clock is what Linux uses for most purposes while it's running.

Most desktop OSs, such as Windows and pre-X versions of Mac OS, set their clocks to the local time. This approach is simple and convenient for people who are used to dealing mainly with local time; however, for purposes of networking, it's inadequate. When it's 4:00 in New York, it's 1:00 in Los Angeles, so network protocols that rely even partly on time can become confused (or at the very least, create confusing log entries) when they operate across time zones. Linux, like other Unix-like OSs, sets its clock to *Coordinated Universal Time (UTC),* which for most purposes is identical to *Greenwich Mean Time (GMT)*—the time in Greenwich, England, unadjusted for Daylight Saving Time. This approach means that Linux systems in New York and Los Angeles (and London and Moscow and Beijing) should have identical times, assuming all are set correctly. For communicating with users, though, these systems need to know their

time zones. For instance, when you type `ls -l` to see a file listing complete with time stamps, Linux reads the time stamp in UTC and then adds or subtracts the appropriate amount of time so that the time stamp appears in your local time. Of course, all of this means that you must be able to set the computer's time zone. On most systems, this is done at system installation; the distribution's installer asks you for your time zone and sets things up correctly. If you erred during installation or if you need to change the time zone for any reason, the upcoming section, "Setting the Time Zone," describes how to fix things.

Linux's internal use of UTC can complicate setting the hardware clock. Ideally, the hardware clock should be set to UTC; however, if your system multi-boots between Linux and an OS that expects the hardware clock to be in local time, you'll have to set the hardware clock to local time and configure Linux to deal with this fact. For the most part, this configuration works well, but you may have to watch the clock the first time you reboot in the spring or fall after changing your clocks because of a Daylight Saving Time adjustment. Depending on your Linux and other OS's settings, your hardware clock might be reset in a way one OS or the other doesn't expect.

Both the hardware clock and the software clock are notoriously unreliable on standard *x*86 hardware; both clocks tend to drift, so your clock can easily end up being several minutes off the correct time within a month or two of being set. To deal with this problem, Linux supports various network protocols for setting the time. The most popular of these is the *Network Time Protocol (NTP)*, which is described in the upcoming section "Using NTP."

Setting the Time Zone

Linux looks to the `/etc/localtime` file for information on its local time zone. (This file is called `/etc/timezone` on some systems, so look for both names.) This file is one of the rare configuration files that's *not* a plain-text file, so you shouldn't try editing it with a text editor. This file could be a file of its own or it could be a symbolic or hard link to another file. If it's a symbolic link, you should be able to determine your time zone by performing a long file listing to see the name of the file to which `localtime` links:

```
$ ls -l /etc/localtime
lrwxrwxrwx  1 root root 36 May 14  2004 /etc/localtime ->
➥ /usr/share/zoneinfo/America/New_York
```

If `/etc/localtime` is a regular file and not a symbolic link, or if you want further confirmation of your time zone, try using the `date` command by itself:

```
$ date
Fri Apr 15 14:17:56 EDT 2005
```

The result includes a standard three-letter time zone code (`EDT` in this example). Of course, you'll need to know these codes, or at least the code for your area. For a list of time zone abbreviations, consult `http://www.timeanddate.com/library/abbreviations/timezones/`. Note that the time zone codes vary depending on whether or not Daylight Saving Time is active, but the Linux time zone files don't change with this detail. In fact, part of what these files do is describe when to change the clock for Daylight Saving Time. If you need to change your time

zone, you should copy or link a sample file from a standard directory location to the `/etc/localtime` file:

1. Log in as `root` or acquire `root` privileges.
2. Change to the `/etc` directory.
3. View the contents of the `/usr/share/zoneinfo` directory. This directory contains files for certain time zones named after the zones or the regions to which they apply, such as `GMT`, `Poland`, and `Japan`. Most users will need to look in subdirectories, such as `/usr/share/zoneinfo/US` for the United States or `/usr/share/zoneinfo/America` for North and South America. These subdirectories contain zone files named after the regions or cities to which they apply, such as `Eastern` or `Los_Angeles`. (The `US` subdirectory contains files named after time zones or states, whereas the `America` subdirectory holds files named after cities.) Identify the file for your time zone. Note that you might use a zone file named after a city other than the one in which you reside but that's in the same time zone as you. For instance, the `New_York` file works fine if you're in Boston, Philadelphia, Cincinnati, Detroit, or any other city in the same (Eastern) time zone as New York.
4. If a `localtime` file exists in `/etc`, delete it or rename it. (For instance, type **`rm localtime`**.)
5. Create a symbolic link from your chosen time zone file to the `/etc/localtime` file. For instance, you might type **`ln -s /usr/share/zoneinfo/US/Eastern localtime`** to set up a computer in the U.S. Eastern time zone. Alternatively, you can copy a file (`cp`) rather than create a symbolic link (`ln -s`). If `/etc` and your target file are on the same filesystem, you could create a hard link rather than a symbolic link if you like.

At this point, your system should be configured to use the time zone you've selected. If you changed time zones, you should be able to see the difference by typing **`date`**, as described earlier. The time zone code on your system should change compared to issuing this command before you changed the `/etc/localtime` file or link. The time should also change by the number of hours between the time zones you've selected (give or take a bit for the time it took you to change the time zone files).

Some distributions provide text-mode or GUI tools to help make time zone changes. Look for a program called `tzsetup`, `tzselect`, or something similar. Typically, these programs ask you for your location in several steps (starting with your continent, then your nation, and perhaps then your state or city) and create an appropriate symbolic link.

Manually Setting the Time

With the time zone set, you can set your system's clock—or more precisely, its *clocks,* because as noted earlier, Linux maintains two clocks: the hardware clock and the software clock. The main tool to set the software clock is `date`, which has the following syntax when setting the clock:

```
date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]
```

Used without any options, this command displays the current date, as described in the previous section. If you pass a time to the program, it sets the clock to that time. This format contains a

month, a day, an hour, and a minute at a minimum, all in two-digit codes (*MMDDhhmm*). You can optionally add a 2- or 4-digit year and the seconds within a minute if you like. You should specify the time in a 24-hour format. For instance, to set the time to 3:02 PM on October 27, 2005, you'd type the following command:

```
# date 102715022005
```

By default, date assumes you're specifying the time in local time. If you want to set the clock in UTC, include the -u, --utc, or --universal option.

Because *x*86 hardware maintains both software and hardware clocks, Linux provides tools to synchronize the two. Specifically, the hwclock utility enables you to set the hardware clock from the software clock or vice versa, as well as do a few other things. Its syntax is fairly straightforward:

```
hwclock [options]
```

You can specify options to accomplish several goals:

Show the hardware clock   To view the hardware clock, pass the -r or --show option. The time is displayed in local time, even if the hardware clock is set to UTC.

Set the hardware clock manually   To set the hardware clock to a date you specify, you need two options: --set and --date=*newdate*. The *newdate* is in the date format that the date program accepts.

Set the hardware clock based on the software clock   If you've set the software clock, you can synchronize the hardware clock to the same value with the --systohc option.

Set the software clock based on the hardware clock   If your hardware clock is accurate but your software clock isn't, you can use the --hctosys option to set the software clock to the hardware clock's value. This option is often used in a SysV startup script to set the system clock when the computer first boots.

Specify UTC or local time   You can tell Linux to treat the hardware clock as storing UTC by using the --utc option or to treat it as holding local time by using the --localtime option. The default is whichever was last used when the hardware clock was set.

Ordinarily, you won't use hwclock directly very often. You might need to use it after a Daylight Saving Time shift if you maintain your hardware clock in local time, but some distributions include scripts that manage this task automatically. You might also want to use it once in a while to keep the hardware clock from drifting too far from an accurate time, but again, some distributions do this automatically as part of the system shutdown procedure.

You can also set the hardware clock via your computer's BIOS setup utility. Consult your motherboard or computer hardware manual for details. You must reboot the system to do this, typically pressing the Delete or some other key at a critical time. You must then find the time option and set it appropriately. If Linux is using UTC, remember to set the clock to UTC rather than local time.

Using NTP

Typically, a clock on an isolated computer needn't be set with any great precision. It doesn't really matter if the time is off by a few seconds, or even a few minutes, so long as the time is reasonably consistent on that one computer for the purpose of `cron`, other scheduling tools, and time stamps. Sometimes, though, maintaining a truly accurate system time is important. This is true for a few scientific, business, or industrial applications (such as astronomical measurements or a system that determines the start and stop times for television broadcasts). In a networked environment, maintaining the correct time can be more important. Time stamps on files might become confused if a file server and its clients have different times, for instance. Worse, a few protocols, such as the Kerberos security suite, embed time stamps in their packets and rely on those time stamps for normal system functioning. If two systems using Kerberos have wildly different times, they might not be able to communicate. For these reasons, several protocols exist to synchronize the clocks of multiple systems. Of these, NTP is the most popular and flexible, so I describe it. You should first understand the basic principles of NTP operation. You can then go on to configuring an NTP server for your network and setting up other systems as NTP clients.

NTP Basics

One of the most popular, flexible, and accurate network time tools is NTP. This protocol creates a tiered hierarchy of time sources, as illustrated in Figure 8.1. At the top of the structure are one or more highly accurate time sources—typically atomic clocks or radio receivers that pull their times from broadcast time signals based on atomic clocks. These are referred to as *stratum 0* time servers, but they aren't directly accessible to any but the *stratum 1* time servers to which they're connected. These stratum 1 computers run NTP servers that deliver the time to *stratum 2* servers, which deliver the time to *stratum 3* servers, and so on for an arbitrary number of strata.

> **NOTE** Other time-setting protocols include one built into the Server Message Block/Common Internet File System (SMB/CIFS) used for Windows file sharing and implemented in Linux by Samba and a protocol used by the `rdate` utility in Linux.

The key to NTP is the fact that each server can deliver time to an expanding number of clients. For instance, if a stratum 1 server has 1,000 clients, each of which has 1,000 clients, and so on, stratum 3 will consist of 1,000,000 systems and stratum 4 will contain 1,000,000,000 systems. Each increase in the stratum number slightly decreases the accuracy of the time signal, but not by much; even a stratum 4 system's clock should be accurate to well under a second, which is accurate enough for almost all purposes. More important, if you run a network, you can set aside one computer as an NTP server and set all your other computers' clocks from that one server. Even if your primary NTP server's clock is off by a second, all the clocks on your network should be set to within a tiny fraction of each other, which is the most important consideration for time-dependent network protocols such as Kerberos.

FIGURE 8.1   NTP enables an expanding pyramid of computers to set their clocks to a highly accurate source signal.
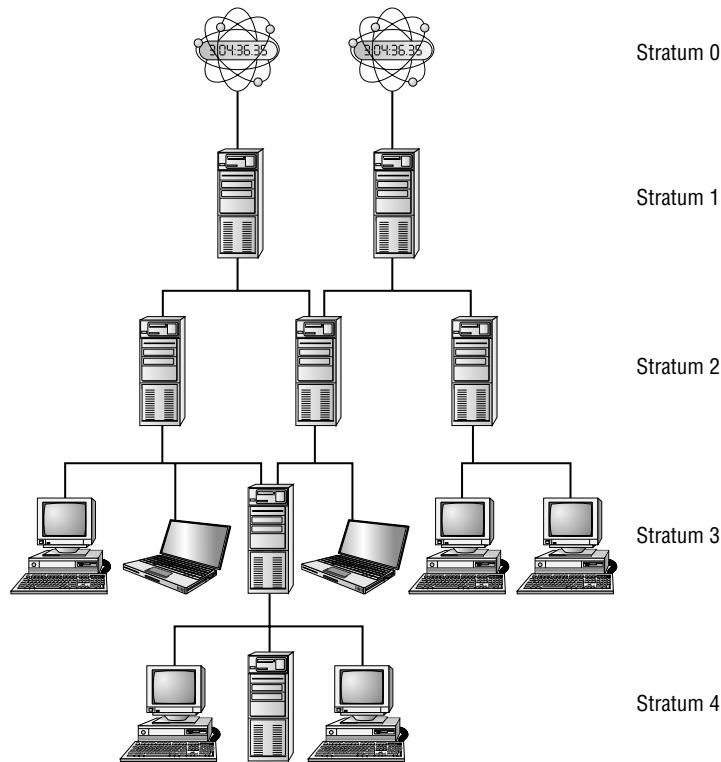
NTP works by measuring the round-trip time for packets between the server and the client. The two systems exchange packets with embedded time stamps and the client then adjusts its time so that it will be synchronized with the source's time stamp but adds a bit to the time reported by the source to account for this round-trip delay. For this reason, when you select an NTP source (as described next, in "Locating a Time Source"), you should pick one with the shortest possible network time delay, all other things being equal. (In truth, several measures of reliability exist, and the NTP programs try to take them all into account.)

The main Linux NTP server program functions as both a server and a client; it sets its clock based on the time of the server to which it's pointed, and it enables other systems to set their clocks based on its own. Even the end points in the NTP hierarchy (the stratum 4 and some stratum 3 servers in Figure 8.1) often run the full NTP server package. The reason is that this software runs constantly and can monitor for and adjust the clock drift that's common in x86 and other computers' clocks, resulting in much more consistent timekeeping than is possible with a program that simply sets the clock and then ignores it until the next time it's run. In other words, NTP doesn't just reset the system clock periodically; the server improves the accuracy of the system clock. In part, this is done through the ntp.drift file, which is usually buried in /var/lib/ntp, but is

sometimes stored in `/etc`. This file holds information on the software clock's inaccuracies, and so can be used to correct for them. A full NTP server, even when it's functioning only as an NTP client, periodically checks with its source systems to keep the system time set correctly and to update the `ntp.drift` file.

Locating a Time Source

You might think that locating an NTP server with a low stratum number (such as stratum 1) is ideal. Although it's true that your own system will have a minutely more accurate clock when using such a source, the best approach in most cases is to synchronize with a stratum 2 or lower system. The reason is that this practice will help keep the load on the stratum 1 servers low, thus improving the overall performance of the NTP network as a whole. An exception might be if you're configuring an NTP server that will itself deliver the time to hundreds or more computers.

To locate an NTP server, you should consult one or more of several sources:

Your ISP   Many Internet service providers (ISPs), including business networks and universities, operate NTP servers for the benefit of their users. These servers are usually very close to your own in a network sense, making them good choices for NTP. You should consult your ISP or the networking department at your organization to learn if such a system is available.

Your distribution's NTP server   Some Linux distributions operate NTP servers for their users. If you happen to be close to these servers in a network sense, they can be good choices; however, chances are this isn't the case, so you might want to look elsewhere.

Public NTP server lists   Lists of public NTP servers are maintained at `http://ntp.isc.org/bin/view/Servers/WebHome`. These servers can be good choices, but you'll need to locate the one closest to you in a network sense, and perhaps contact the site you choose to obtain permission to use it.

> **NOTE** The closest server in a network sense might not be the closest computer in a geographic sense. For instance, a national ISP might route all traffic through just one or two hub sites. The result can be that traffic from, say, Atlanta Georgia to Tampa Florida might go through Chicago Illinois. Such a detour is likely to increase round-trip time and decrease the accuracy of NTP. In such a situation, a user in Atlanta might be better off using a Chicago NTP server than one in Tampa, even though Tampa is much closer geographically.

Once you've located a few possible time servers, try using `ping` to determine the round-trip time for packets to this system. If any systems have very high `ping` times, you may want to remove them from consideration.

Configuring NTP Servers

When setting up a network to use NTP, select one system (or perhaps two for a network with several dozen or more computers) to function as the primary NTP server. This computer needn't be a very powerful one, but it must have always-up access to the network. You can then install the NTP server and configure it.

Most Linux distributions ship the NTP software in a package called `ntp`, `xntp`, `ntpd`, or `xntpd`. Look for this package and, if it's not already installed, install it. If you can't find this package, check http://ntp.isc.org/bin/view/Main/SoftwareDownloads. This site hosts NTP source code, which you can compile and install as described in Chapter 2. Alternatively, you can look for a binary package for another distribution. Either way, if you don't install your distribution's own NTP package, you'll need to create your own SysV startup script or start the NTP daemon in some other way.

Once NTP is installed, look for its configuration file, `/etc/ntp.conf`. This file contains various NTP options, but the most important are the server lines:

```
server clock.example.com
server ntp.pangaea.edu
server time.luna.edu
```

Each of these lines points to a single NTP server. When your local NTP daemon starts up, it will contact all the servers specified in `/etc/ntp.conf`, measure their accuracy against each other, and settle on one as its primary time source. Typically, you list about three upstream time servers for a system that's to serve many other computers. This practice enables your server to weed out any servers that deliver a bad time signal, and it also gives automatic fallback in case an upstream server becomes temporarily or permanently unavailable. If your NTP server won't be serving many computers itself, you might want to configure it for three servers initially and then drop the ones your system isn't using as its primary time source after a day or two. This will reduce the load on these servers.

You may want to peruse your configuration file for entries you might want to remove. For instance, the configuration file might contain references to servers you'd rather not use or other odd options with associated comments that make you think they're inappropriate. Generally speaking, you shouldn't adjust other entries in the `ntp.conf` file, but of course special circumstances or odd starting files may require you to make changes.

Once you've made your changes, start or restart your NTP daemon. Typically, this is done via a SysV startup script:

```
# /etc/init.d/ntpd restart
```

You may need to change the path to the file, the SysV script filename, or the option (change `restart` to `start` if you're starting NTP for the first time). Most distributions configure NTP to start whenever the system boots once you install the server. Consult Chapter 6 for details of changing this configuration.

To verify that NTP is working, you can use the `ntpq` program, which is an interactive program that accepts various commands. Figure 8.2 shows it in operation, displaying the output of the `peers` command, which displays the servers to which your NTP server is connected. In Figure 8.2, three external servers are listed, plus `LOCAL(0)`, which is the last-resort reference source of the computer's own clock. The `refid` column shows the server to which each system is synchronized, the `st` column shows the stratum of the server, and additional columns show more technical information. The server to which yours is synchronized is denoted by an asterisk (`*`), other servers with good times are indicated by plus signs (`+`), and most other symbols (such as `x` and `-`) denote servers that have been discarded from consideration for various reasons. Consult `ntpq`'s `man` page for more information on its operation.

FIGURE 8.2   The ntpq program enables you to verify that an NTP server is functioning correctly.


```
$ ntpq
ntpq> peers
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
+clock.example.c 128.227.205.3    2 u  186 1024  377   23.016   -0.624  10.787
*ntp.pangaea.edu 172.17.1.243     2 u  269 1024  377   41.464    4.139   2.553
+time.luna.edu   18.145.0.30      2 u  186 1024  377   20.183   -0.293   0.366
 LOCAL(0)        LOCAL(0)        10 l   23   64  377    0.000    0.000   0.001
ntpq>
```

> **NOTE**   You won't see a server selected as the source until a few minutes after you restart the NTP daemon. The reason is that your local NTP process takes a while to determine which of the sources is providing the best signal.

Configuring NTP Clients

Once you've configured one or more NTP servers, you can configure the rest of your systems to point to them. Their configuration is done just like the NTP server configuration, with a couple of exceptions:

- You set your NTP clients to refer to the NTP server (or servers) you've just configured rather than to an outside NTP source. This way, your local systems won't put an additional burden on the outside NTP server you've selected.

- You may want to ensure that your NTP clients can't be accessed as servers. This is a security measure. You can do this with an `iptables` firewall rule, as described in Chapter 7 (block port 123), or by using the `restrict default ignore` line in `ntp.conf`. This line tells the server to ignore all incoming NTP requests. Ideally, you should use both methods.

Once you've configured a client, restart its NTP daemon. You can then use `ntpq` to check its status. You should see that it refers only to your network's own NTP server or servers. These systems should be listed as belonging to a stratum with a number one higher than the servers to which they refer.

In some cases, a simpler way to set the time on a client is to use `ntpdate`. This program is part of the NTP suite, and it performs a one-time clock setting. To use it, type the command name followed by the hostname or IP address of an NTP server:

```
# ntpdate clock.example.com
```

Some NTP packages include a call to `ntpdate` in their NTP daemon startup scripts in order to ensure that the system is set to the correct time when it starts. The `ntpdate` command, however, has been deprecated and could disappear from the NTP package at any time. Instead, you can start `ntpd` with its `-g` option, which enables it to perform a one-time clock setting to a value that's wildly divergent from the current time. (Ordinarily, `ntpd` exits if the time server's time differs from the local time by more than a few minutes.)

🌐 Real World Scenario

Serving Time to Windows Systems

If your network hosts both Linux and Windows systems, you may want to use a Linux system as a time source for Windows clients or conceivably even use a Windows server as a time source for Linux clients. One way to do this is to run NTP on Windows. Consult `http://geodsoft.com/howto/timesync/wininstall.htm` or perform a Web search to locate NTP software for Windows systems. For Windows NT/200*x*/XP, you can type `NET TIME /SETSNTP:`*`time.server`*, where *`time.server`* is the name of your local NTP time server. This command performs a one-time setting of the clock but doesn't run in the background as the full NTP package does on Linux. Running this command in a Windows login script may be adequate for your purposes.

For older Windows 9*x*/Me systems, you can type `NET TIME \\`*`SERVER`*` /SET /YES` to have the system set the time to the time maintained by *SERVER*, which must be a Windows or Samba file or print server. This command doesn't use NTP, but if you've got a Linux system that runs both NTP and Samba, it can be a good way to get the job done.

Running Jobs in the Future

Some system maintenance tasks should be performed at regular intervals and are highly automated. For instance, the `/tmp` directory (which holds temporary files created by many users) tends to collect useless data files. Linux provides a means of scheduling tasks to run at specified times to handle such issues. This tool is the `cron` program, which runs what are known as *cron jobs*. A related tool is `at`, which enables you to run a command on a one-time basis at a specified point in the future as opposed to doing so on a regular basis, as `cron` does.

The Role of *cron*

The `cron` program is a daemon, so it runs continuously, looking for events that cause it to spring into action. Unlike most daemons, which are network servers, `cron` responds to temporal events. Specifically, it "wakes up" once a minute, examines configuration files in the `/var/spool/cron` and `/etc/cron.d` directories and the `/etc/crontab` file, and executes commands specified by these configuration files if the time matches the time listed in the files.

There are two types of `cron` jobs: *system cron jobs* and *user cron jobs*. System `cron` jobs are run as `root` and perform system-wide maintenance tasks. By default, most Linux distributions include system `cron` jobs that clean out old files from `/tmp`, perform log rotation (as described earlier, in "Rotating Log Files"), and so on. You can add to this repertoire, as described shortly. Ordinary users can create user `cron` jobs, which might run some user program on a regular basis. You can also create a user `cron` job as `root`, which might be handy if you need to perform some task at a time not supported by the system `cron` jobs, which are scheduled rather rigidly.

One of the critical points to remember about `cron` jobs is that they run unsupervised. Therefore, you shouldn't call any program in a `cron` job if that program requires user input. For instance, you wouldn't run a text editor in a `cron` job. You might, however, run a script that automatically manipulates text files, such as log files.

Creating System *cron* Jobs

The `/etc/crontab` file controls system `cron` jobs. This file normally begins with several lines that set environment variables, such as `PATH` and `MAILTO` (the former sets the path, and the latter is the address to which programs' output is mailed). The file then contains several lines that resemble the following:

```
02 4 * * * root run-parts /etc/cron.daily
```

This line begins with five fields that specify the time. The fields are, in order, the minute (0–59), the hour (0–23), the day of the month (1–31), the month (1–12), and the day of the week (0–7; both 0 and 7 correspond to Sunday). For the month and day of the week values, you can use the first three letters of the name rather than a number, if you like.

In all cases, you can specify multiple values in several ways:

- An asterisk (`*`) matches all possible values.
- A list separated by commas (such as `0,6,12,18`) matches any of the specified values.
- Two values separated by a dash (`-`) indicate a range, inclusive of the end points. For instance, `9-17` in the hour field specifies a time of from 9:00 AM to 5:00 PM
- A slash, when used in conjunction with some other multi-value option, specifies stepped values—a range in which some members are skipped. For instance, `*/10` in the minute field indicates a job that's run every 10 minutes.

After the first five fields, `/etc/crontab` entries continue with the account name to be used when executing the program (`root` in the preceding example) and the command to be run (`run-parts /etc/cron.daily` in this example). The default `/etc/crontab` entries generally use `run-parts`, `cronloop`, or a similar utility that runs any executable scripts within a directory. Thus, the preceding example runs all the scripts in `/etc/cron.daily` at 4:02 AM every day. Most distributions include monthly, daily, weekly, and hourly system `cron` jobs, each corresponding to scripts in a directory called `/etc/cron.interval`, where *interval* is a word associated with the run frequency. Others place these scripts in `/etc/cron.d/interval` directories.

The exact times chosen for system `cron` jobs to execute vary from one distribution to another. Normally, though, daily and longer-interval `cron` jobs run early in the morning—between midnight and 6:00 AM. Check your `/etc/crontab` file to determine when your system `cron` jobs run.

To create a new system cron job, you may create a script to perform the task you want performed (as described in Chapter 6) and copy that script to the appropriate /etc/cron.*interval* directory. When the runtime next rolls around, cron will run the script.

> **TIP** Before submitting a script as a cron job, test it thoroughly. This is particularly important if the cron job will run when you're not around. You don't want a bug in your cron job script to cause problems by filling the hard disk with useless files or producing thousands of e-mail messages when you're not present to quickly correct the problem.

If you need to run a cron job at a time or interval that's not supported by the standard /etc/ crontab, you can either modify that file to change or add the cron job runtime or create a user cron job, as described shortly. If you choose to modify the system cron job facility, model your changes after an existing entry, changing the times and script storage directory as required.

> **WARNING** System cron job storage directories should be owned by root, and only root should be able to write to them. If ordinary users can write to a system cron directory, unscrupulous users could write scripts to give themselves superuser privileges and place them in the system cron directory. The next time cron runs those scripts, the users will have full administrative access to the system.

Creating User *cron* Jobs

To create a user cron job, you use the crontab utility, not to be confused with the /etc/ crontab configuration file. The syntax for crontab is as follows:

```
crontab [-u user] [-l | -e | -r] [file]
```

If given without the -u *user* parameter, crontab modifies the cron job associated with the current user. (User cron jobs are often called crontabs, but with the word already used in reference to the system-wide configuration file and the utility itself, this usage can be perplexing.) The crontab utility can become confused by the use of su to change the current user identity, so if you use this command, it's safest to also use -u *user*, even when you are modifying your own cron job.

If you want to work directly on a cron job, use one of the -l, -e, or -r options. The -l option causes crontab to display the current cron job; -r removes the current cron job; and -e opens an editor so that you can edit the current cron job. (Vi is the default editor, but you can change this by setting the VISUAL or EDITOR environment variables, as described in Chapter 1.)

Alternatively, you can create a cron job configuration file and pass the filename to crontab using the *file* parameter. For instance, **crontab -u tbaker my-cron** causes crontab to use my-cron for tbaker's cron jobs.

Whether you create the cron job and submit it via the *file* parameter or edit it via -e, the format of the cron file is similar to that described earlier. You can set environment variables by using the form *VARIABLE=value*, or you can specify a command preceded by five numbers or wildcards to

indicate when the job is to run. In a user `cron` job, however, you do *not* specify the username used to execute the job, as you do with system `cron` jobs. That information is derived from the owner of the `cron` job. Listing 8.2 shows a sample `cron` job file. This file runs two programs at different intervals: The `fetchmail` program runs every 30 minutes (on the hour and half hour), and `clean-adouble` runs on Mondays at 2:00 AM. Both programs are specified via complete paths, but you could include a PATH environment variable and omit the complete path specifications.

Listing 8.2: A Sample User *cron* Job File

```
SHELL=/bin/bash
MAILTO=tbaker
HOME=/home/tbaker
0,30 * * * * /usr/bin/fetchmail -s
0 2 * * mon /usr/local/bin/clean-adouble $HOME
```

Ultimately, user `cron` job files are stored in the `/var/spool/cron/tabs` directory. (Some distributions use `crontabs` rather than `tabs` in this path.) Each file in this directory is named after the user under whose name it will run, so for example, `tbaker`'s file will be called `/var/spool/cron/tabs/tbaker`. You shouldn't directly edit the files in this directory, though; instead, use `crontab` to make changes.

Access to the `cron` facility may be restricted in several ways:

Executable permissions The permissions on the `cron` and `crontab` programs may be restricted using standard Linux permissions mechanisms, as described in Chapter 4. Not all distributions configure themselves in this way, but for those that do, users who should be able to schedule jobs using `cron` should be added to the appropriate group. This group is often called `cron`, but you should check the group owner and permissions on the `/usr/sbin/cron` and `/usr/bin/crontab` program files to be sure.

Allowed users list The `/etc/cron.allow` file contains a list of users who should be permitted access to `cron`. If this file is present, only users whose names appear in the file may use `cron`; all others are denied access. If this file is not present, anybody may use `cron`, assuming access isn't restricted by executable permissions or a disallowed users list.

Disallowed users list The `/etc/cron.deny` file contains a list of users who should be denied access to `cron`. If this file is present, any user whose name appears in the file is denied access to `cron`, but all others may use it, assuming executable permissions and the allowed users list don't restrict access.

Using *anacron*

Although `cron` is a great tool for performing certain tasks, such as rotating log files, on systems that are up most or all of the time, it's a much less useful tool on systems that are frequently shut down, such as notebook computers or even many desktop systems. Frequently, late-night `cron` jobs are never executed on such systems, which can lead to bloated log files, cluttered `/tmp` directories, and other problems.

EXERCISE 8.2

Creating User *cron* Jobs

cron jobs can be a useful way to run programs at regular times. In this exercise, you'll create a simple user cron job that will mail you the output of an ifconfig command on a daily basis. This exercise assumes that you're authorized to use cron as an ordinary user. To configure your cron job, follow these steps:

1. Log into the Linux system as a normal user.

2. Launch an xterm from the desktop environment's menu system, if you used a GUI login method.

3. Create and edit a file called cronjob in your home directory. Use your favorite text editor for this purpose. The file should contain the following lines:

```
SHELL=/bin/bash
MAILTO=yourusername
00 12 * * * /sbin/ifconfig
```

   Be sure to type these lines exactly; a typo will cause problems. One exception: Substitute your e-mail address on the Linux system or elsewhere for *yourusername*; cron uses the MAILTO environment variable to determine to whom to e-mail the output of cron jobs.

4. Type **crontab cronjob** to install the cronjob file as a cron job. Note that this command replaces any existing user cron jobs that might exist. If you've already defined user cron jobs for your account, you should edit your existing cronjob file to add the line calling ifconfig rather than create a new file.

5. Wait for 12:00 noon (00 12 in the cron time format). When this time rolls around, you should have a new e-mail waiting for you with the contents of the ifconfig output.

Instead of waiting for 12:00 noon, you can substitute a time that's a couple of minutes in the future. Remember that cron specifies minutes first, followed by the hour in a 24-hour format. For instance, if you create the file at 3:52 PM, you might enter 54 15 as the first two numbers on the final line of the file; this will cause the cron job to execute at 15:54 on a 24-hour clock, or 3:54 PM.

One solution to such problems is anacron (http://anacron.sourceforge.net). This program is designed as a supplement to cron to ensure that regular maintenance jobs are executed at reasonable intervals. It works by keeping a record of programs it should execute and how frequently it should do so, in days. Whenever anacron is run, it checks to see when it last executed each of the programs it's configured to manage. If a period greater than the program's execution interval has passed, anacron runs the program. Typically, anacron itself is run from a system startup script, and perhaps from a cron job. You can then reconfigure your

regular system `cron` jobs as `anacron` jobs and be sure they'll execute even on systems that are regularly shut down for long stretches of time.

Like `cron`, `anacron` is controlled through a configuration file named after itself: `/etc/anacrontab`. This file consists of three main types of lines: comment lines (denoted by a leading hash mark, #), environment variable assignments (as in SHELL=/bin/bash), and job definition lines. This last type of line contains four fields:

```
period  delay  identifier  command
```

The *period* is how frequently, in days, the command should be run. The *delay* is a delay period, in minutes, between the time `anacron` starts and the time the command is run, if it should be run. This feature is intended to help keep the system from being overloaded if `anacron` determines it needs to run many commands when it starts up. The *identifier* is a string that identifies the command. You can pass it to `anacron` on the command line to have `anacron` check and, if necessary, run only that one command. Finally, *command* is the command to be run. This is a single command or script name, optionally followed by any parameters it might take.

Listing 8.3 shows a sample `/etc/anacrontab` file. This file sets a couple of environment variables; PATH is particularly important if any scripts call programs without specifying their complete paths. The three job definition lines tell `anacron` to run the `run-parts` command, passing it the name of a different directory for each line. This command is used on some distributions to run `cron` jobs, so the effect of calling it from `anacron` is to take over `cron`'s duties. The first line, run once a day, causes `anacron` to run (via `run-parts`) the scripts in `/etc/cron.daily`; the second line causes the scripts in `/etc/cron.weekly` to be run once a week; and the third, run once every 30 days, runs the scripts in `/etc/cron.monthly`.

Listing 8.3: Sample *etc/anacrontab* File

```
SHELL=/bin/bash
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
# format: period delay job-identifier command
1       5       cron.daily      run-parts /etc/cron.daily
7       10      cron.weekly     run-parts /etc/cron.weekly
30      15      cron.monthly    run-parts /etc/cron.monthly
```

Of course, to do any good, `anacron` must be called itself. This is typically done in one of two ways:

Via a startup script   You can create a startup script to run `anacron`. A simple SysV startup script that takes no options but that runs `anacron` should do the job if configured to run from your regular runlevel. Alternatively, you could place a call to `anacron` in a local startup script, such as Fedora and Red Hat's `/etc/rc.d/rc.local` or SuSE's `/etc/boot.d/boot.local`.

Via a `cron` job   You can create a `cron` job to run `anacron`. Typically, this call will replace your regular system `cron` job entries (in `/etc/crontab`), and you'll probably want to call `anacron` on a daily basis or more frequently.

The startup script approach is best employed on systems that are shut down and started up frequently, such as laptops or desktop systems that are regularly shut down at the end of the day. One drawback to this approach is that it can cause sluggish performance when the system is booted if anacron needs to run a time-consuming task. Calling anacron via a cron job can shift the burden to off hours, but if cron can reliably run anacron, cron could as easily and reliably run the jobs that anacron runs. Typically, you'd use a cron job if the system is sometimes, but not always, left running overnight. This will ensure that anacron and the jobs it handles are run fairly frequently, if not on a completely regular basis. Alternatively, you could call anacron more frequently than once a day. For instance, if it's called once every six hours, it will almost certainly be called during a typical eight-hour workday.

> For a desktop system, you might try calling anacron via a cron job at the user's typical lunch break. This will help minimize the disruption caused by any resource-intensive programs that anacron must run.

No matter how you run anacron, you should be sure to disable any cron jobs that anacron now handles. If you don't do so, those tasks will be performed twice, which may needlessly burden your system. Because anacron measures its run intervals in days, it's not a useful utility for running hourly cron jobs. Thus, you shouldn't eliminate any hourly system cron jobs when you edit your cron configuration for anacron.

Using *at*

Sometimes cron and anacron are overkill. You might simply want to run a single command at a specific point in the future on a one-time basis rather than on an ongoing basis. For this task, Linux provides another command: at. In ordinary use, this command takes a single option (although options to fine-tune its behavior are also available): a time. This time can take any of several forms:

Time of day   You can specify the time of day as *HH:MM*, optionally followed by AM or PM if you use a 12-hour format. If the specified time has already passed, the operation is scheduled for the next occurrence of that time—that is, for the next day.

noon, midnight, or teatime   These three keywords stand for what you'd expect (teatime is 4:00 PM).

Day specification   To schedule an at job more than 24 hours in advance, you must add a day specification after the time of day specification. This can be done in numeric form, using the formats *MMDDYY*, *MM/DD/YY*, or *DD.MM.YY*. Alternatively you can specify the date as *month-name day* or *month-name day year*.

A specified period in the future   You can specify a time using the keyword now, a plus sign (+), and a time period, as in now + 2 hours to run a job in two hours.

> The at command relies on a daemon, atd, to be running. If your system doesn't start atd automatically, you may need to configure a SysV startup script to do so.

When you run `at` and give it a time specification, the program responds with its own prompt, `at>`, which you can treat much like your normal `bash` or other command shell prompt. When you're done typing commands, press Ctrl+D to terminate input. Alternatively, you can pass a file with commands by using the `-f` parameter to `at`, as in **`at -f commands.txt noon`** to use the contents of `commands.txt` as the commands you want to run at noon.

The `at` command has several support tools. The most important of these is `atd`, the `at` daemon. This program must be running for `at` to do its work. If it's not, check for its presence using `ps`. If it's not running, look for a SysV startup script and ensure that it's enabled, as described in Chapter 6.

Other `at` support programs include `atq`, which lists pending `at` jobs; `atrm`, which removes an `at` job from the queue; and `batch`, which works much like `at` but executes jobs when the system load level drops below 0.8. These utilities are all fairly simple. To use **`atq`**, simply type its name. (The program does support a couple of options, but chances are you won't need them; consult `atq`'s `man` page for details.) To use `atrm`, type the program name and the number of the `at` job, as returned by `atq`. For instance, you might type **`atrm 12`** to remove `at` job number 12.

The `at` facility supports access restrictions similar to those of `cron`. Specifically, the `/etc/at.allow` and `/etc/at.deny` files work analogously to the `/etc/cron.allow` and `/etc/cron.deny` files. There are a few wrinkles with `at`, though. Specifically, if neither `at.allow` nor `at.deny` exists, only `root` may use `at`. If `at.allow` exists, the users it lists are granted access to `at`; if `at.deny` exists, everybody *except* those mentioned in this file is granted access to `at`. This differs from `cron`, in which everybody is granted access if neither access-control file is present. This tighter default security on `at` means that the program is seldom installed with restrictive execute permissions, but of course you could use program file permissions to deny ordinary users the ability to run `at` if you want an extra layer of security.

Backing Up the System

Many things can go wrong on a computer that might cause it to lose data. Hard disks can fail, you might accidentally enter some extremely destructive command, a cracker might break into your system, or a user might accidentally delete a file, to name just a few possibilities. To protect against such problems, it's important that you maintain good backups of the computer. To do this, select appropriate backup hardware, choose a backup program, and implement backups on a regular schedule. You should also have a plan in place to recover some or all of your data should the need arise.

Common Backup Hardware

Just about any device that can store computer data and read it back can be used as a backup medium. The best backup devices are inexpensive, fast, high in capacity, and reliable. They don't usually need to be *random access* devices, though. Random access devices are capable of quickly accessing any piece of data. Hard disks, floppy disks, and CD-ROMs are all random access devices. These devices contrast with *sequential access* devices, which must read through

all intervening data before accessing the sought-after component. Tapes are the most common sequential-access devices.

Traditionally, tapes have been the backup medium of choice. Although they're sequential-access devices, they're inexpensive on a per-gigabyte basis, and tapes have usually had capacities that match or exceed the capacities of typical computer hard disks. In the past few years, though, the price gap between tapes and other media—particularly recordable DVDs and hard disks—has been narrowing. Thus, these devices are now often used for backups. Typically, recordable DVDs (or the lower-capacity CD-R or CD-RW media) are used for backing up basic system installations or particular projects, while hard disks mounted in special removable drive bays are used for complete system backups. Tapes can fill either role.

> If you restrict computers' main installation partitions to about 1 to 1.4GB, those entire partitions will most likely fit, when compressed, on standard 700MB CD-Rs. This can simplify backup and recovery efforts.

It's generally wise to keep multiple backups and to store some of them away from the computers they're meant to protect. Such offsite storage protects your data in case of fire, vandalism, or other major physical traumas. Keeping several backups makes it more likely you'll be able to recover something, even if it's an older backup, should your most recent backup medium fail.

If you decide to use a tape drive, your choices aren't over. Several competing tape formats are in common use. These include Travan, which dominates the low end of the spectrum; digital audio tape (DAT), which is generally considered a step up; digital linear tape (DLT) and Super DLT, which are well respected for use on servers and networks; 8mm, which is similar to DAT but has higher capacities; and Advanced Intelligent Tape (AIT), which is a high-end tape medium. Each of these competes at least partially with some of the others. Travan drives tend to be quite inexpensive (typically $200–$500), but the media are pricey. The other formats feature more costly drives ($500–$4000 for a single drive), but the media cost less. Maximum capacities vary: under 1GB for obsolete forms of Travan, 20GB for top-of-the-line Travan, and 160GB for the largest Super DLT drives. Overall, Travan is a good solution for low-end workstations; DAT is best used on high-end workstations, small servers, and small networks; and the other formats are all good for high-end workstations, servers, and networks.

If you decide to use hard disks in removable mounts as a backup medium, you'll need ordinary internal drives and mounting hardware. The hardware comes in two parts: a mounting bay that fits in the computer and a frame in which you mount the hard drive. To use the system, you slide the frame with hard drive into the mounting bay. You can get by with one of each component, but it's best to buy one frame for each hard drive, which effectively raises the media cost. From a Linux software point of view, removable hard disk systems work like regular hard disks or other removable disk systems, like Zip disks. Most of these systems use ATA disks, which you'll access as `/dev/hdb`, `/dev/hdc`, or some other ATA device identifier. The disks are likely to be partitioned, and the partitions are likely to hold ordinary Linux filesystems.

Optical media (recordable DVDs, CD-Rs, and CD-RWs) are odd from a software interface perspective: You must use tools such as `cdrecord` to write to them rather than accessing their device files directly. CD-R and CD-RW drives and media are very well standardized, both from

a software and a hardware perspective, so buy these based on price and features. Several competing recordable DVD formats exist, though, and the market is constantly changing. If you need the higher capacity of a recordable DVD, you should research the current market and formats before buying. (You should be able to use `cdrecord` or similar Linux programs to write to any recordable DVD drive, though.)

Common Backup Programs

Linux supports several backup programs. Some are tools designed to back up individual files, directories, or computers. Others build on these simpler tools to provide network backup facilities. Basic backup programs include `tar`, `cpio`, and `dump`. In some cases, using `dd` to back up a small partition can make sense as well.

The *tar* Utility

The `tar` program's name stands for "tape archiver." Despite this fact, `tar` can be used to back up data to other media. In fact, *tarballs* (archive files created by tar and typically compressed with `gzip` or `bzip2`) are often used for transferring multiple files between computers in one step, such as when distributing source code.

The `tar` program is a complex package with many options. Most of what you'll do with the utility, however, can be covered with a few common commands. Table 8.1 lists the primary `tar` commands, and Table 8.2 lists the qualifiers that modify what the commands do. Whenever you run `tar`, you use exactly one command and you usually use at least one qualifier.

TABLE 8.1   tar Commands

Command	Abbreviation	Description
`--create`	c	Creates an archive
`--concatenate`	A	Appends `tar` files to an archive
`--append`	r	Appends non-tar files to an archive
`--update`	u	Appends files that are newer than those in an archive
`--diff` or `--compare`	d	Compares an archive to files on disk
`--list`	t	Lists archive contents
`--extract` or `--get`	x	Extracts files from an archive

TABLE 8.2   tar Qualifiers

Command	Abbreviation	Description
`--directory` *dir*	C	Changes to directory *dir* before performing operations
`--file [`*host*`:]`*file*	f	Uses file called *file* on computer called *host* as the archive file
`--listed-incremental` *file*	g	Performs incremental backup or restore, using *file* as a list of previously archived files
`--one-file-system`	l	Backs up or restores only one filesystem (partition)
`--multi-volume`	M	Creates or extracts a multitape archive
`--tape-length` *N*	L	Changes tapes after *N* kilobytes
`--same-permissions`	p	Preserves all protection information
`--absolute-paths`	P	Retains the leading / on filenames
`--verbose`	v	Lists all files read or extracted; when used with `--list`, displays file sizes, ownership, and time stamps
`--verify`	W	Verifies the archive after writing it
`--exclude` *file*	(none)	Excludes *file* from the archive
`--exclude-from` *file*	X	Excludes files listed in *file* from the archive
`--gzip` or `--ungzip`	z	Processes archive through `gzip`
`--bzip2`	j (some older versions used I or y)	Processes archive through bzip2

Of the commands listed in Table 8.1, the most commonly used are `--create`, `--extract`, and `--list`. The most useful qualifiers from Table 8.2 are `--file`, `--listed-incremental`, `--one-file-system`, `--same-permissions`, `--gzip`, `--bzip2`, and `--verbose`. (`--bzip2` is a fairly recent addition, so it may not work if you're using an older version of tar.) If you fail to specify a filename with the `--file` qualifier, tar will attempt to use a default device, which is often (but not always) a tape device file.

The upcoming section "Backing Up a Computer" describes how to use tar in practice.

The *cpio* Utility

The cpio program is similar in principle to tar, but the details of its operation differ. As with tar, you can direct its output straight to a tape device. This can be a convenient way to back up the computer because it requires no intermediate storage. To restore data, you use cpio to read directly from the tape device file.

The cpio utility has three operating modes:

Copy-out mode   This mode, activated by use of the -o or --create option, creates an archive and copies files into it.

Copy-in mode   You activate copy-in mode by using the -i or --extract option. This mode extracts data from an existing archive. If you provide a filename or a pattern to match, cpio will extract only the files whose names match the pattern you provide.

Copy-pass mode   This mode is activated by the -p or --pass-through option. It combines the copy-out and copy-in modes, enabling you to copy a directory tree from one location to another.

> The copy-out and copy-in modes are named confusingly.

In addition to the options used to select the mode, cpio accepts many other options, the most important of which are summarized in Table 8.3. To back up a computer, you'll combine the --create (or -o) option with one or more of the options in Table 8.3; to restore data, you'll do the same, but use --extract (or -i). In either case, cpio acts on filenames that you type at the console. In practice, you'll probably use the redirection operator (<) to pass a filename list to the program.

TABLE 8.3   Options for Use with cpio

Option	Abbreviation	Description
--reset-access-time	-a	Resets the access time after reading a file so that it doesn't appear to have been read.
--append	-A	Appends data to an existing archive.
--pattern-file=*filename*	-E *filename*	Uses the contents of *filename* as a list of files to be extracted in copy-in mode.
--file=*filename*	-F *filename*	Uses *filename* as the cpio archive file; if this parameter is omitted, cpio uses standard input or output.

TABLE 8.3 Options for Use with cpio *(continued)*

Option	Abbreviation	Description
--format=*format*	-H *format*	Uses a specified format for the archive file. Common values for *format* include bin (the default, an old binary format), crc (a newer binary format with a checksum), and tar (the format used by tar).
N/A	-I *filename*	Uses the specified *filename* instead of standard input. (Unlike -F, this option does not redirect output data.)
--no-absolute-filenames	N/A	In copy-in mode, extracts files relative to the current directory, even if filenames in the archive contain full directory paths.
N/A	-O *filename*	Uses the specified *filename* instead of standard output. (Unlike -F, this option does not redirect input data.)
--list	-t	Displays a table of contents for the input.
--unconditional	-u	Replaces all files without first asking for verification.
--verbose	-v	Displays filenames as they're added to or extracted from the archive. When used with -t, displays additional listing information (similar to ls -l).

The *dump* and *restore* Utilities

One traditional Unix backup tool is called dump. This program operates on a somewhat lower level than tar and cpio; where these programs read and write files like most programs, dump uses low-level filesystem access to back up and restore data on the inode level. One result of this difference is that a dump utility is tied to the filesystem it services. The main Linux dump utility works only with ext2 and ext3 filesystems; it can't be used to back up ReiserFS, XFS, JFS, or any other type of filesystem. XFS provides its own dump utility (xfsdump), but ReiserFS and JFS lack dump utilities.

Quite a few dump options exist, but a basic backup uses the -f *file* option to specify a backup file and passes the files or directories to be backed up:

dump -f /dev/st0 /home

This command backs up the /home directory tree to /dev/st0 (a SCSI tape drive). For more details on dump options, consult its man page.

The dump utility, despite the fact that it doesn't work on many Linux filesystems, holds a special place in Linux because the next-to-the-last entry in /etc/fstab lines tells dump whether it should automatically back up a filesystem when called in certain ways. Chapter 4 describes /etc/fstab in more detail.

To restore data from a dump backup, you must use the restore program. Like dump, restore takes a large number of options, so consult its man page for details. Be aware that you can restore files only to the same filesystem type on which they were originally stored—for instance, you can't restore files from an ext2 or ext3 filesystem to an XFS partition.

As a general rule, the dump and restore utilities should not be used because they aren't reliable on 2.4.*x* and later kernels. (Linus Torvalds himself has cautioned against its use; see http://lwn.net/2001/0503/a/lt-dump.php3.)

The *dd* Utility

A backup program that operates on an even lower level than dump is dd. This program is a low-level copying program, and when it's given the device file for a partition as input, it copies that partition's contents to the output file you specify. This output file could be another partition identifier, a tape device, or a regular file, to name three possibilities. The input and output files are passed with the if=*file* and of=*file* options:

```
# dd if=/dev/hda3 of=/dev/st0
```

This command backs up the /dev/hda3 disk partition to /dev/st0 (a SCSI tape drive). The result is a very low-level backup of the partition that can be restored by swapping the if= and of= options:

```
# dd if=/dev/st0 of=/dev/hda3
```

The dd utility can be a good way to create exact backups of entire partitions, but as a general backup tool, it has serious problems. It backs up the *entire* partition, including any empty space. For instance, a 2GB partition that holds just 5MB of files will require 2GB of storage space. Restoring individual files is also impossible unless the target device is a random access device that can be mounted; if you back up to tape, you must restore everything (at least to a temporary file or partition) to recover a single file. Finally, you can't easily restore data to a partition that's smaller than the original partition, and when restoring to a larger partition, you'll end up wasting some of the space available on that partition.

Despite these problems, dd can be handy in some situations. It can be a good way to make an exact copy of a floppy disk, for instance, including its boot sector. You can use dd to copy a disk for which Linux lacks filesystem drivers. For instance, you can back up and restore a Windows New Technology File System (NTFS) partition using dd. If you need to create multiple identical Linux installations, you can do so by using dd to copy a working installation to multiple computers, so long as they've got hard disks of the same size.

Backing Up a Computer

The `tar` command is generally considered the lowest common denominator backup program. Tapes created with `tar` can be read on non-Linux systems, and all mainstream Linux distributions ship with `tar`.

On the downside, `cpio` and `tar` have a compression problem: These programs don't compress data themselves. To do this, these programs rely on an external program, such as `gzip` or `bzip2`, to compress an entire `cpio` or `tar` archive. The problem with this approach is that if an error occurs while restoring the compressed archive, all the data from that error onward will be lost. This makes compressed `cpio` or `tar` archives risky for backup. Fortunately, most tape drives support compression in their hardware, and these use more robust compression algorithms. Therefore, if your tape drive supports compression, you should *not* compress a `cpio` or `tar` backup. Let the tape drive do that job, and if there's a read error at restore, you'll probably lose just one or two files. If your tape drive doesn't include built-in compression features, you should either not compress your backups or use another utility, most of which don't suffer from this problem.

To back up a computer with `tar`, you provide a list of files or directories on the command line, along with the `--create` (`c`) command and several qualifiers:

```
# tar cvlpf /dev/st0 /home / /boot /var
```

This command lists directories in a particular order. Because tape is a sequential-access medium, the system will restore items in the order in which they were backed up. Therefore, for the fastest partial restores, list the filesystems that you most expect to have to restore first. In this example, /home is listed first because users sometimes delete files accidentally. Backing up /home first, therefore, results in quicker restoration of such files.

The `--verbose` (`v`) option displays a list of files as they're backed up. Ordinarily, `tar` descends the directory tree, including mounted filesystems; the `--one-file-system` (`l`) option prevents `tar` from descending into filesystems mounted within the specified directory tree. The `--same-permissions` (`p`) option ensures that all permission information is backed up. The `--file` (`f`) option is required to specify the output device (`/dev/st0` in this example).

After creating a backup with `tar`, you may want to use the `tar --diff` (also known as `--compare`, or `d`) command to verify the backup you've just written against the files on disk. Alternatively, you can include the `--verify` (`W`) qualifier to have this done automatically. Verifying your backup doesn't guarantee it will be readable when you need it, but it should at least catch major errors caused by severely degraded tapes. On the other hand, the verification will almost certainly return a few spurious errors because of files whose contents have legitimately changed between being written and being compared. This may be true of log files, for instance.

Planning a Backup Schedule

Regular computer backup is important, but precisely *how* regularly is a matter that varies from one system to another. If a computer's contents almost never change (as might be true of a dedicated router or a workstation whose user files reside on a file server), backups once a month or

---

Real World Scenario

Backing Up Using Optical Media

Optical media require special backup procedures. Normally, cdrecord accepts input from a program like mkisofs, which creates an ISO-9660 filesystem—the type of filesystem that's most often found on CD-ROMs.

One option for backing up to optical discs is to use mkisofs and then cdrecord to copy files to the disc. If you copy files "raw" in this way, though, you'll lose some information, such as write permission bits. You'll have better luck if you create a tar (or other backup program) file on disk, much as you would when you back up to tape. You would then use mkisofs to place that archive in an ISO-9660 filesystem, and then you would burn the ISO-9660 image file to the optical disc. The result will be a CD-R that you can mount and that will contain an archive you can read with tar.

A somewhat more direct option is to create an archive file and burn it directly to the optical disc using cdrecord, bypassing mkisofs. Such a disc won't be mountable in the usual way, but you can access the archive directly by using the CD-ROM device file. On restoration, this works much like a tape restore except that you specify the CD-ROM device filename (such as /dev/cdrom) instead of the tape device filename (such as /dev/st0).

---

even less often might be in order. For critical file servers, once a day is not too often. You'll have to decide for yourself just how frequently your systems require backup. Take into consideration factors such as how often the data change, the importance of the data, the cost of recovering the data without a current backup, and the cost of making a backup. Costs may be measured in money, your own time, users' lost productivity, and perhaps lost sales.

Even the most zealous backup advocate must admit that creating a full backup of a big system on a regular basis can be a tedious chore. A backup can easily take several hours, depending on backup size and hardware speed. For this reason, most backup packages, including tar, support *incremental backups*. You can create these using the --listed-incremental file qualifier to tar, as shown in this example:

```
# tar cvplf /dev/st0 --listed-incremental /root/inc / /home
```

This command stores a list of the files that have been backed up (along with identifying information to help tar determine when the files have changed) in /root/inc. The next time the same command is issued, tar will not back up files that have already been backed up; it will back up only new files. Thus, you can create a schedule in which you do a full backup of the entire computer only occasionally—say, once a week or once a month. You'd do this by deleting the increment file and running a backup as usual. On intervening weeks or days, you can perform an incremental backup, in which only new and changed files are backed up. These incremental backups will take comparatively little time.

Performing incremental backups has a couple of drawbacks. One is that they complicate restoration. Suppose you do a full backup on Monday and incremental backups every other day. If a system fails on Friday, you'll need to restore the full backup and several incremental backups. Second, after restoring an incremental backup, your system will contain files that you'd deleted since the full backup. If files have short life spans on a computer, this can result in a lot of "dead" files being restored when the time comes to do so.

Despite these problems, incremental backups can be an extremely useful tool for helping make backups manageable. They can also reduce wear and tear on tapes and tape drives, and they can minimize the time it takes to restore files if you know that the files you need to restore were backed up on an incremental tape.

> **WARNING**   Whether you perform incremental backups or nothing but complete backups, you should maintain multiple backups. Murphy's Law guarantees that your backup will fail when you need it most, so having a backup for your backup (even if it's from a week or a month earlier) can help immensely. A typical backup plan includes a rotating set of backup tapes. For instance, you might have two tapes per week—one for a full backup on one day and one to hold several incremental backups. Eight tapes will then hold backups for four weeks.

Preparing for Disaster: Backup Recovery

Creating backups is advisable, but doing this isn't enough. You must also have some way to restore backups in case of disaster. This task involves two aspects: partial restores and emergency recovery.

Partial restores involve recovering just a few noncritical files. For instance, users might come to you and ask you to restore files from their home directories. You can do so fairly easily by using the --extract (x) tar command, as in the following lines:

```
# cd /
# tar xvlpf /dev/st0 home/username/filename
```

> **NOTE**   This sequence involves changing to the root directory and issuing a relative path to the file or directory that must be restored. This is required because tar normally strips away the leading / in files it backs up, so the files are recorded in the archive as relative filenames. If you try to restore a file with an absolute filename, it won't work.

With most backup programs, you'll need to know the exact name of the file or directory you want to restore in order to do this. If you don't know the exact filename, you may need to use the --list (t) command to tar to examine the entire contents of the tape, or at least everything until you see the file you want to restore.

> If you use incremental backups, you can use the incremental file list to locate the name of the file you want to restore.

A much more serious problem is that of recovering a system that's badly damaged. If your hard disk has crashed or your system has been invaded by crackers, you must restore the entire system from scratch, without the benefit of your normal installation. You can take any of several approaches to this problem, including the following:

Distribution's installation disk   Most Linux distributions' installation disks have some sort of emergency recovery system. These may come as separate boot floppy images or as options to type during the boot process. In any event, these images are typically small but functional Linux systems with a handful of vital tools, such as `fdisk`, `mkfs`, Vi, and `tar`. Check your distribution's documentation or boot its boot media and study its options to learn more.

CD-based Linux system   Several Linux systems are now available that boot from CD-ROM. One example is Knoppix (`http://www.knoppix.com`); another is a demo version of SuSE (`http://www.suse.com`; but the site is being transitioned to Novell's site, `http://www.novell.com`). Both of these systems can be used to help recover or restore a corrupted Linux installation.

Emergency system on removable disk   You can create your own emergency system on a removable disk. If you have a moderately high-capacity removable disk, like a Zip or LS-120 disk, you can create a moderately comfortable Linux system on this disk. The ZipSlack distribution (a variant of Slackware, `http://www.slackware.com`) is particularly handy for this purpose because it's designed to fit on a 100MB Zip disk. You can use this even if your regular installation is of another version of Linux.

Emergency recovery partition   If you plan ahead, you might create a small emergency installation of your preferred distribution alongside the regular installation. You should *not* automatically mount this system in `/etc/fstab`. This system can be useful for recovering from some problems, like software filesystem corruption, but it's not useful for others, like a total hard disk failure.

Partial reinstallation   You can reinstall a minimal Linux system and then use it to recover your original installation. This approach is much like the emergency recovery partition approach, but it takes more time at disaster recovery. On the other hand, it will work even if your hard disk is completely destroyed.

Whatever approach you choose to use, you should test it before you need it. Learn at least the basics of the tools available in any system you plan to use. If you use unusual backup tools (such as commercial backup software), be sure to copy those tools to your emergency system or have them available on a separate floppy disk. If you'll need to recover clients via network links, test those setups as well.

You may not be able to *completely* test your emergency restore tools. Ideally, you should boot the tools, restore a system, and test that the system works. This may be possible if you have spare hardware on which to experiment, but if you lack this luxury, you may have to make do with performing a test restore of a few files and testing an emergency boot procedure—say,

using `LOADLIN` (a DOS-based boot loader that can boot a Linux system when LILO or GRUB isn't installed or working). Note that a freshly restored system will not be bootable; you'll need a kernel on a DOS boot floppy and `LOADLIN`, or some other emergency boot system, to boot the first time. You can then reinstall LILO or GRUB to restore the system's ability to boot from the hard disk.

Summary

Routine system administration involves a variety of tasks, many of which center around user management. Adding, deleting, and modifying user accounts and groups are critical tasks that all system administrators must master. Also related to users, you should know where to go to modify the default user environment.

System log files are critical troubleshooting tools that are maintained by the system. You should be able to configure what data is logged to what files and know how to use these log files.

Time management is important in Linux. Setting the Linux clocks (both hardware and software) and configuring NTP to keep the software clock accurate are important tasks. Tools that rely on the time include `cron`, `anacron`, and `at`, which enable the system to run programs in the future. These tools are used for many common system tasks, including rotating log files.

Finally, backing up Linux is an important but often overlooked duty. Without backups, data that took weeks, months, or even years to generate could be lost in an instant. Various backup tools, such as `tar` and `cpio`, are useful for handling backups, but you must be able to design a backup schedule to make backups. Restoring data is also important, so you should have an emergency recovery plan at the ready.

Exam Essentials

Summarize methods of creating and modifying user accounts. Accounts can be created or modified with the help of tools designed for the purpose, such as `useradd` and `usermod`. Alternatively, you can directly edit the `/etc/passwd` and `/etc/shadow` files, which hold the account information.

Describe the function of groups in Linux. Linux groups enable security features to be applied to arbitrary groups of users. Each group holds an arbitrary collection of users, and group permissions can be set on files giving all group members the same access rights to the files.

Explain the purpose of the skeleton files. Skeleton files provide a core set of configuration files that should be present in users' home directories when those directories are created. They provide a starting point for users to modify their important shell and other configuration files.

Summarize how to configure system logging. System logging is controlled via the `/etc/syslog.conf` file. Lines in this file describe what types of log data, generated by programs, are sent to log files and to which log files the log messages should go.

Describe how log rotation is managed.   Log rotation is controlled via the `/etc/logrotate`
`.conf` file (which typically refers to files in `/etc/logrotate.d`). Entries in these files tell the
system whether to rotate logs at fixed intervals or when they reach particular sizes. When a log
rotates, it's renamed (and possibly compressed), a new log file is created, and the oldest archived
log file may be deleted.

Explain the two types of clocks in *x*86 hardware.   The hardware clock retains time when the
system is powered down, but it isn't used by most programs while the system is running. Such pro-
grams refer to the software clock, which is set from the hardware clock when the computer boots.

Summarize the function of NTP   The Network Time Protocol (NTP) enables a computer to
set its clock based on the time maintained by an NTP server system. NTP can function as a tiered
protocol, enabling one system to function as a client to an NTP server and as a server to addi-
tional NTP clients. This structure enables a single highly accurate time source to be used by any-
where from a few to (theoretically) billions of computers via a tiered system of links.

Describe the difference between `cron` and `anacron`.   The `cron` utility is Linux's traditional
tool for running programs in the future. It enables you to schedule jobs to run at specific repeat-
ing times, but it requires the computer to be up at that time for the job to run. The `anacron` util-
ity runs jobs at repeating times, but jobs may be run late if the computer is powered down or
the utility isn't turned on, making `anacron` a better choice for laptops or desktop systems that
are regularly turned off.

Explain the difference between system and user `cron` jobs.   System `cron` jobs are controlled
from `/etc/crontab`, are created by `root`, and may be run as any user (but most commonly as
`root`). Systen `cron` jobs are typically run at certain fixed times on an hourly, daily, weekly, or
monthly basis. User `cron` jobs may be created by any user (various security measures permit-
ting), are run under the authority of the account with which they're associated, and may be run
at just about any repeating interval desired.

Summarize the common Linux backup programs.   The `tar` and `cpio` programs are both file-
based backup tools that create archives of files using ordinary file access commands. The `dump`
program backs up files on an inode basis, which ties `dump` to just one filesystem. The `dd` program
is a file-copy program, but when it's fed a partition device file, it will copy the entire partition
on a very low-level basis, which is useful for creating low-level image backups of Linux or non-
Linux filesystems.

Describe some approaches to restoring a complete Linux system from backup.   Complete
restores require careful preparation and planning before an emergency occurs. You may use
your distribution's installation media, a bootable CD-ROM distribution such as Knoppix, a
small-disk distribution such as ZipSlack, a small prepared emergency partition, or a partial
installation of a regular distribution to perform the restore.

Review Questions

1. Which of the following are legal Linux usernames? (Select all that apply.)

   A. `larrythemoose`

   B. `4sale`

   C. `PamJones`

   D. `Samuel_Bernard_Delaney_the_Fourth`

2. Why are groups important to the Linux user administration and security models?

   A. They can be used to provide a set of users with access to files without giving *all* users access to the files.

   B. They allow you to set a single login password for all users within a defined group.

   C. Users may assign file ownership to a group, thereby hiding their own creation of the file.

   D. By deleting a group, you can quickly remove the accounts for all users in the group.

3. An administrator types **`chage -M 7 time`**. What is the effect of this command?

   A. The `time` account's password must be changed at least once every seven days.

   B. All users must change their passwords at least once every seven days.

   C. All users are permitted to change their passwords at most seven times.

   D. The `time` account's age is set to seven months.

4. What is wrong with the following `/etc/passwd` file entry? (Select all that apply.)

   `4sally:x:529:Sally Jones:/home/myhome:/bin/passwd`

   A. The default shell is set to `/bin/passwd`, which is an invalid shell.

   B. The username is invalid; Linux usernames can't begin with a number.

   C. The home directory doesn't match the username.

   D. Either the UID or the GID field is missing.

5. You want `sally`, `tom`, and `dale` to be members of the group `managers` (GID 501). How would you edit the `managers` entry in `/etc/group` to accomplish this goal?

   A. `managers:501:sally tom dale`

   B. `managers:501:sally:tom:dale`

   C. `managers:x:501:sally:tom:dale`

   D. `managers:x:501:dale,sally,tom`

6. What types of files might be reasonable files to include in `/etc/skel`? (Select all that apply.)

   A. A copy of the `/etc/shadow` file

   B. An empty set of directories to encourage good file management practices

   C. A `README` or similar welcome file for new users

   D. A starting `.bashrc` file

7. Which of the following system logging codes represents the *highest* priority?

   A. emerg

   B. warning

   C. crit

   D. debug

8. Which of the following configuration files does the logrotate program consult for its settings?

   A. /etc/logrotate.conf

   B. /usr/sbin/logrotate/logrotate.conf

   C. /usr/src/logrotate/logrotate.conf

   D. /etc/logrotate/.conf

9. Your manager has asked that you configure logrotate to run on a regular, unattended basis. What utility/feature should you configure to make this possible?

   A. at

   B. logrotate.d

   C. cron

   D. inittab

10. You're configuring a Linux system that does not boot any other OS. What is the recommended time to which the computer's hardware clock should be set?

    A. Helsinki time

    B. Local time

    C. US Pacific time

    D. UTC

11. What does the following command and its output reveal about the computer?

    ```
    $ ls -l /etc/localtime
    lrwxrwxrwx  1 root root 30 Apr 15 14:24 /etc/localtime ->
    ➥ /usr/share/zoneinfo/US/Eastern
    ```

    A. The time zone is misconfigured; there is no /usr/share/zoneinfo/US/Eastern time zone file.

    B. The computer's security could easily be breached; this file should not have rwxrwxrwx permissions.

    C. The computer is configured to use the U.S. Eastern time zone as the local time.

    D. The current time in the U.S. Eastern time zone is 14:24 (2:24 PM).

12. You've set your system (software) clock on a Linux-only computer to the correct time, and now you want to set the hardware clock to match. What command might you type to accomplish this goal?

A. `date --sethwclock`

B. `ntpdate`

C. `hwclock --utc --systohc`

D. `time --set --hw`

13. You've configured one computer on your five-computer network, `gateway.pangaea.edu`, as an NTP server that obtains its time signal from `ntp.example.com`. What computer(s) should your network's other computers use as their time source(s)?

A. You should consult a public NTP server list to locate the best server for you.

B. Both `gateway.pangaea.edu` and `ntp.example.com`.

C. Only `ntp.example.com`.

D. Only `gateway.pangaea.edu`.

14. Which of the following tasks is likely to be handled by a `cron` job? (Select all that apply.)

A. Starting an important server when the computer boots

B. Finding and deleting old temporary files

C. Scripting supervised account creation

D. Monitoring the status of servers and e-mailing a report to the superuser

15. Which of the following lines, if used in a user `cron` job, will run `/usr/local/bin/cleanup` twice a day?

A. `15 7,19 * * * tbaker /usr/local/bin/cleanup`

B. `15 7,19 * * * /usr/local/bin/cleanup`

C. `15 */2 * * * tbaker /usr/local/bin/cleanup`

D. `15 */2 * * * /usr/local/bin/cleanup`

16. You're installing Linux on a laptop computer. Which of the following programs might you want to add to ensure that log rotation is handled correctly?

A. `tempus`

B. `anacron`

C. `crontab`

D. `ntpd`

17. What do the following commands accomplish? (The administrator presses Ctrl+D after typing the second command.)

```
# at teatime
at> /usr/local/bin/system-maintenance
```

A. Nothing; these commands aren't valid.

B. Nothing; `teatime` isn't a valid option to `at`.

C. Nothing; you may only type valid `bash` built-in commands at the `at>` prompt.

D. The /usr/local/bin/system-maintenance program or script is run at 4:00 PM.

18. Which of the following commands are commonly used to back up Linux systems? (Select all that apply.)

A. `restore`

B. `tar`

C. `tape`

D. `cpio`

19. You need to restore some files that were accidentally deleted. Which of the following commands can be used to list the contents of an archive stored on a SCSI tape prior to actually restoring the files?

A. `tar uvf /dev/st0`

B. `tar cvf /dev/st0`

C. `tar xvf /dev/st0`

D. `tar tvf /dev/st0`

20. You arrive at work on Monday morning to find that the server has crashed. All indications point to the crash as occurring after midnight on Monday morning. Scripts automatically do a full backup of the server every Friday night and an incremental backup all other nights. Which tapes do you need to restore the data on a new server? (Select all that apply.)

A. Thursday's tape

B. Friday's tape

C. Saturday's tape

D. Sunday's tape

Answers to Review Questions

1. A, C. A Linux username must contain fewer than 32 characters and start with a letter, and it may consist of letters, numbers, and certain symbols. Options A and C both meet these criteria. (Option C uses mixed upper- and lowercase characters, which is legal but discouraged.) Option B begins with a number, which is invalid. Option D is longer than 32 characters.

2. A. Groups provide a good method of file-access control. Although they may have passwords, these are *not* account login passwords; those passwords are set on a per-account basis. Files do have associated groups, but these are *in addition* to individual file ownership and so they cannot be used to mask the file's owner. Deleting a group *does not* delete all the accounts associated with the group.

3. A. The chage command changes various account expiration options. The −M parameter sets the maximum number of days for which a password is valid, and in the context of the given command, time is a username. Thus, option A is correct. Options B, C, and D are all simply made up.

4. B, D. As stated in option B, Linux usernames may not begin with numbers, so the username is invalid. The /etc/passwd entries have third and fourth fields of the UID and the GID, but this line has only one of those fields (which one is intended is impossible to determine); this example line's fourth field is clearly the fifth field of a valid entry. Option A is incorrect because, although /bin/passwd is an unorthodox login shell, it's perfectly valid. This configuration might be used on, say, a Samba file server or a POP mail server to enable users to change their passwords via SSH without granting login shell access. Option C is a correct observation but an incorrect answer; the username and the user's home directory name need not match.

5. D. Option D shows a valid /etc/group entry that has the desired effect. (Note that the order of users in the comma-separated user list is unimportant.) Option A has two problems: It's missing a password field (x in the correct entry) and the usernames are separated by spaces rather than commas. Option B also has two problems: It's missing a password field and its usernames are separated by colons rather than commas. Option C has just one problem: Its usernames are separated by colons rather than commas.

6. B, C, D. Files in /etc/skel are copied from this directory to new users' home directories by certain account-creation tools. Thus, files you want in all new users' home directories should reside in /etc/skel. Options B, C, and D all describe reasonable possibilities, although none is absolutely required. Including a copy of /etc/shadow in /etc/skel would be a very bad idea, though, because this would give all users access to all other users' encrypted passwords, at least as of the moment of account creation.

7. A. The emerg priority code is the highest code available and so is higher than all the other options. (The panic code is equivalent to emerg but isn't one of the options.) From highest to lowest priorities, the codes given as options are emerg, crit, warning, and debug.

8. A. The logrotate program consults a configuration file called /etc/logrotate.conf, which includes several default settings and typically refers to files in /etc/logrotate.d to handle specific log files.

9. C. The `logrotate` program can be started automatically—and unattended—on a regular basis by adding an entry for it in `cron`. The `at` utility would be used if you wanted the program to run only once, while `logrotate.d` defines how the program is to handle specific log files. The `inittab` file is used for services and startup and not for individual programs.

10. D. Linux, like Unix, maintains its time internally in Coordinated Universal Time (UTC), so setting the computer's hardware clock to UTC is the recommended procedure for computers that run only Linux. Although Linus Torvalds spent time at the University of Helsinki, Helsinki time (as in option A) has no special place in Linux. Local time (as in option B) is appropriate if the computer dual-boots to an OS, such as Windows, that requires the hardware clock to be set to local time, but this is the second-best option for a Linux-only system. Option C's US Pacific time, like Helsinki time, has no special significance in Linux.

11. C. The `/etc/localtime` file holds time zone information and is frequently a symbolic link pointing to a time zone file in the `/usr/share/zoneinfo` directory tree. The `US/Eastern` file in this directory configures the system for the U.S. Eastern time zone, so option C is correct. This file is a common one, so option A is incorrect. Although the permissions on the link do look alarming at first glance, this is not a problem for symbolic links, which always have these permissions, so option B is incorrect. Although it's possible that the computer's time happens to be 2:24 PM, the time in the listing is the time the `/etc/localtime` link was created, not the current time, so option D is incorrect.

12. C. The `hwclock` utility is used to view or set the hardware clock. The `--utc` option tells it to use UTC, which is appropriate for a Linux-only system, while `--systohc` sets the hardware clock based on the current value of the software clock. Thus, option C is correct. Option A's `date` utility can be used to set the software clock but not the hardware clock; it has no `--sethwclock` option. Option B's `ntpdate` is used to set the software clock to the time maintained by an NTP server; it doesn't directly set the hardware clock. Option D's `time` command is used to time how long a command takes to complete; it has no `--set` or `--hw` options and does not set the hardware clock.

13. D. Once you've configured one computer on your network to use an outside time source and run NTP, the rest of your computers should use the first computer as their time reference. This practice reduces the load on the external time servers, as well as your own external network traffic. Thus, option D is correct. (Very large networks might configure two or three internal time servers that refer to outside servers for redundancy, but this isn't necessary for the small network described in the question.) Option A describes the procedure to locate a time server for the first computer configured (`gateway.pangaea.edu`) but not for subsequent computers. Although configuring other computers to use `ntp.example.com` instead of or in addition to `gateway.pangaea.edu` is possible, doing so will needlessly increase your network traffic and the load on the `ntp.example.com` server.

14. Answers: B, D. The `cron` utility is a good tool for performing tasks that can be done in an unsupervised manner, like deleting old temporary files or checking to see that servers are running correctly. Tasks that require interaction, like creating accounts, are not good candidates for `cron` jobs, which must execute unsupervised. Although a `cron` job could restart a crashed server, it's not normally used to start a server when the system boots; that's done through SysV startup scripts or a super server.

15. B. User cron jobs don't include a username specification (tbaker in options A and C). The */2 specification for the hour in options C and D causes the job to execute every other hour; the 7,19 specification in options A and B causes it to execute twice a day, on the 7th and 19th hours (in conjunction with the 15 minute specification, that means at 7:15 AM and 7:15 PM).

16. B. The anacron program is a supplement to cron that helps ensure that log rotation, /tmp directory cleanup, and other traditional cron tasks are handled even when the computer is shut down (and, hence, when cron isn't running) for extended periods of time. Thus, this is the program to add to the system to achieve the stated goal. There is no common Linux utility called tempus. Option C's crontab is the name of a file or program for controlling cron, which is likely to be an unreliable means of log rotation on a laptop computer. The ntpd program is the NTP daemon, which helps keep the system clock in sync with an external source. Although running ntpd on a laptop computer is possible, it won't directly help with the task of scheduling log rotation.

17. D. The at command runs a specified program at the stated time in the future. This time may be specified in several ways, one of which is teatime, which stands for 4:00 PM. Thus, option D is correct. The objections stated in options A, B, and C are all invalid.

18. B, D. The tar and cpio programs are common Linux archive-creation utilities that are frequently used for backing up the computer. The restore command restores (but does not back up) data; its backup counterpart command is dump, but dump is no longer a recommended backup tool on Linux. There is no standard tape command in Linux.

19. D. With the tar utility, the --list (t) command is used to read the archive and display its contents. The --verbose (v) option creates a verbose file listing, while --file (f) specifies the filename—/dev/st0 for a SCSI tape device. Option D uses all of these features. Options A, B, and C all substitute other commands for --list, which is required by the question.

20. B, C, D. In order to restore the data, you must restore the most recent full backup—which was done on Friday night. After the full restore, you must restore the incremental backups in the order in which they were done. In this case, two incremenatals (Saturday's and Sunday's) were done after the full backup and they must be restored as well.

Chapter

9

Basic Networking

THE FOLLOWING LINUX PROFESSIONAL INSTITUTE OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **1.107.2 Manage printers and print queues (weight: 1)**
- ✓ **1.107.3 Print files (weight: 1)**
- ✓ **1.107.4 Install and configure local and remote printers (weight: 1)**
- ✓ **1.112.1 Fundamentals of TCP/IP (weight: 4)**
- ✓ **1.112.3 TCP/IP configuration and troubleshooting (weight: 7)**
- ✓ **1.112.4 Configure Linux as a PPP client (weight: 3)**
- ✓ **1.113.1 Configure and manage `inetd`, `xinetd`, and related services (weight: 4)**

Most Linux systems are connected to a network, either as clients or as servers (and often as both). For this reason, understanding how to configure Linux's basic networking tools is necessary for fully configuring Linux. To begin with this task, you must first understand the basics of modern networking, such as the nature of network addresses and the types of tools that are commonly used on networks. From there, you can move on to Linux network configuration, both for local networks and using a modem to connect Linux to the Internet. Part of this task is configuring a *super server,* which manages the connections for other servers. (Server configuration generally is covered in more detail in Chapter 10, "Managing Servers.")

You might not think of printing as being a network topic, but in Linux, printing involves network protocols, even when printing locally. For this reason, this chapter also covers printer configuration, including local printer configuration, sharing printers with other computers, and tools used to manage print jobs.

Understanding TCP/IP Networking

Networking involves quite a few components that are built atop one another. These include network hardware, data packets, and protocols for data exchange. Together, these components make up a *network stack*. The most common network stack today is the *Transmission Control Protocol/Internet Protocol (TCP/IP)* stack, but this isn't the only stack available. Nonetheless, understanding the basics of TCP/IP theory will help you to configure and manage networks.

Basic Functions of Network Hardware

Network hardware is designed to enable two or more computers to communicate with one another. Most network hardware comes as a card you plug into a computer, although some devices are external and interface through an ordinary port like a USB port and other network "cards" are built into computer motherboards. Many networks rely on wires or cables to transmit data between machines as electrical impulses, but network protocols that use radio waves or even light to do the job are growing rapidly in popularity.

Sometimes the line between network hardware and peripheral interface ports can be blurry. For instance, a parallel port is normally not considered a network port, but when it is used with the Parallel Line Interface Protocol (PLIP; `http://tldp.org/HOWTO/PLIP.html`), the parallel port becomes a network device. More commonly, a USB or RS-232 serial port can become a network interface when used with the *Point-to-Point Protocol (PPP)*, as described in the upcoming section "Configuring Linux as a PPP Client."

At its core, network hardware is hardware that facilitates the transfer of data between computers. Hardware that's most often used for networking includes features that help this transfer in various ways. For instance, such hardware may include ways to address data intended for specific remote computers, as described later in the section "Hardware Addresses." When basically non-network hardware is pressed into service as a network medium, the lack of such features may limit the utility of the hardware or require extra software to make up for the lack. If extra software is required, you're unlikely to notice the deficiencies as a user or system administrator because the protocol drivers handle the work, but this makes the hardware more difficult to configure and more prone to sluggishness or other problems than other types of network hardware.

Types of Network Hardware

Aside from traditionally non-network ports like USB, RS-232 serial, and parallel ports, Linux supports several types of common network hardware. The most common of these is Ethernet, which comes in several varieties. Most modern Ethernet hardware uses *twisted-pair* cabling, which consists of pairs of wires twisted around each other to minimize interference. Such varieties of Ethernet are identified by a *-T* suffix to the Ethernet variety name, as in 10Base-T or 100Base-T. The numbers denote the speed of the protocol in megabits per second (Mbps). In the late 1990s, 100Base-T took over from 10Base-T as the standard in office and even home networks. More recently, 1000Base-T and Ethernet variants that use optical cabling and that are capable of 1000Mbps speeds (that is, *gigabit Ethernet*) have been growing in popularity.

Other types of network hardware exist, but most are less common than Ethernet. These include Token Ring, LocalTalk, Fiber Distributed Data Interface (FDDI), High-Performance Parallel Interface (HIPPI), and Fibre Channel. Token Ring was common on some IBM-dominated networks in the 1990s but has been steadily losing ground to Ethernet for years. Likewise, LocalTalk was the favored medium for early Macintoshes, but new Macs ship with Ethernet instead of LocalTalk. FDDI, HIPPI, and Fibre Channel are all high-speed interfaces that are used in high-performance applications. Some of these protocols support significantly greater maximum cable lengths than does Ethernet, which makes them suitable for linking buildings that are many yards, or even miles, apart.

Wireless networking is an exception to Ethernet's increasing dominance. Common wireless protocols include 802.11b (aka Wi-Fi), 802.11a, and 802.11g. These protocols support speeds of 11Mbps (for 802.11b) and up. Wireless networking is particularly useful for laptop computers, but it's even handy for homes and small offices that don't have adequate wired network infrastructures in place.

WARNING   If you use a wireless protocol, your data are transmitted via radio waves, which are easily intercepted. Wireless protocols include optional encryption, but this feature is often disabled by default, and it's also notoriously poor. Thus, you should be extremely careful about transferring sensitive data, such as passwords and credit card numbers, over a wireless link. If you must do so, use a strong encryption protocol, such as the Secure Shell (SSH) login tool or Secure Sockets Layer (SSL) encryption on a web page.

In addition to the network cards you place in your computers, you need network hardware outside of the computer. With the exception of wireless networks, you'll need some form of network cabling that's unique to your hardware type. (For 100Base-T Ethernet, get cabling that meets at least Category 5, or Cat-5, specifications.) Many network types, including twisted-pair Ethernet, require the use of a central device known as a *hub* or *switch*. You plug every computer on a local network into this central device, as shown in Figure 9.1. The hub or switch then passes data between the computers.

As a general rule, switches are superior to hubs. Hubs mirror all traffic to all computers, whereas switches are smart enough to send packets only to the intended destination. Switches also allow *full-duplex* transmission, in which both parties can send data at the same time (like two people talking on a telephone). Hubs permit only *half-duplex* transmission, in which the two computers must take turns (like two people using walkie-talkies). The result is that switches let two pairs of computers engage in full-speed data transfers with each other; with a hub, these two transfers would interfere with each other.

Network Packets

Modern networks operate on discrete chunks of data known as *packets*. Suppose you want to send a 100KB file from one computer to another. Rather than send the file in one burst of data, your computer breaks it down into smaller chunks. The system might send 100 packets of 1KB each, for instance. This way, if there's an error sending one packet, the computer can resend just that one packet rather than the entire file. (Many network protocols include error-detection procedures.)

When the recipient system receives packets, it must hold on to them and reassemble them in the correct order to re-create the complete data stream. It's not uncommon for packets to be delayed or even lost in transmission, so error-recovery procedures are critical for protocols that handle large transfers. Some types of error recovery are handled transparently by the networking hardware.

F I G U R E   9 . 1   Many networks link computers together via a central device known as a hub or switch.

Hub or switch
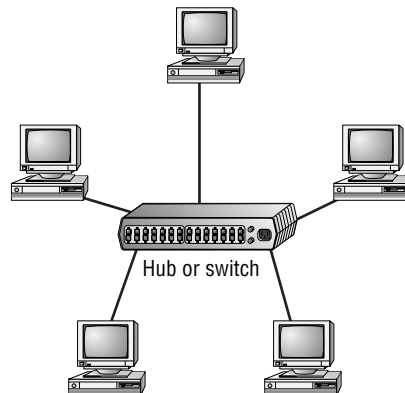
There are several types of packets, and they can be stored within each other. For instance, Ethernet includes its own packet type (known as a *frame*), and the packets generated by networking protocols that run atop Ethernet, such as those described in the next section, are stored within Ethernet frames. All told, a data transfer can involve several layers of wrapping and unwrapping data. With each layer, packets from the layer above may be merged or split up.

Network Protocol Stacks

It's possible to think of network data at various levels of abstractness. For instance, at one level, a network carries data packets for a specific network type (such as Ethernet); the data packets are addressed to specific computers on a local network. Such a description, while useful for understanding a local network, isn't very useful for understanding higher-level network protocols, such as those that handle e-mail transfers. These high-level protocols are typically described in terms of commands sent back and forth between computers, frequently without reference to packets. The addresses used at different levels also vary, as explained in the upcoming section "Types of Network Addresses."

A *protocol stack* is a set of software that converts and encapsulates data between layers of abstraction. For instance, the stack can take the commands of e-mail transfer protocols, and the e-mail messages that are transferred, and package them into packets. Another layer of the stack can take these packets and repackage them into Ethernet frames. There are several layers to any protocol stack, and they interact in highly specified ways. It's often possible to swap out one component for another at any given layer. For instance, at the top of each stack is a program that uses the stack, such as an e-mail client. You can switch from one e-mail client to another without too much difficulty; both rest atop the same stack. Likewise, if you change a network card, you have to change the driver for that card, which constitutes a layer very low in the stack. Applications above that driver can remain the same.

Each computer in a transaction requires a compatible protocol stack. When they communicate, the computers pass data down their respective stacks and then send data to the partner system, which passes the data up its stack. Each layer on the receiving system sees the data as packaged by its counterpart on the sending computer.

Protocol stacks are frequently represented graphically in diagrams like Figure 9.2, which shows the configuration of the TCP/IP protocol stack that dominates the Internet today. As shown in Figure 9.2, client programs at the application layer initiate data transfers. These requests pass through the transport, internet, and link layers on the client computer, whereupon they leave the client system and pass to the server system. (This transfer can involve a lot of complexity not depicted in Figure 9.2.) On the server, the process reverses itself, with the server program running at the application layer replying to the client program. This reply reverses the journey, travelling down the server computer's stack, across the network, and up the stack on the client. A full-fledged network connection can involve many back-and-forth data transfers.

Each component layer of the sending system is equivalent to a layer on the receiving system, but these layers need not be absolutely identical. For instance, you can have different models of network card at the link layer, or you can even use entirely different network hardware types, such as Ethernet and Token Ring, if some intervening system translates between them. The computers may run different OSs entirely and hence use different—but logically equivalent—protocol stacks. What's important is that the stacks operate in compatible ways.

FIGURE 9.2 Information travels "down" and "up" protocol stacks, being checked and packed at each step of the way.
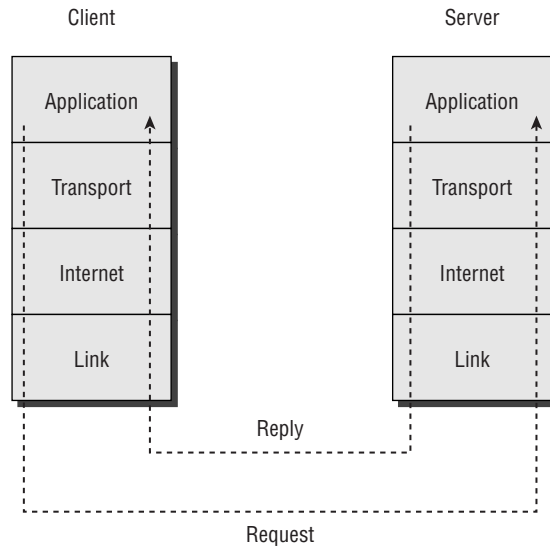
Linux was designed with TCP/IP in mind, and the Internet is built atop TCP/IP. Other protocol stacks are available, though, and you might occasionally run into them. In particular, Net-BEUI was the original Microsoft and IBM protocol stack for Windows, AppleTalk was Apple's initial protocol stack, and the Internet Packet Exchange/Sequenced Packet Exchange (IPX/SPX) was Novell's favored protocol stack. All three are now fading in importance, but you might still run into them. Linux supports AppleTalk and IPX/SPX but not NetBEUI.

TCP/IP Protocol Types

Within TCP/IP, several different protocols exist. Each of these protocols can be classified as falling on one of the four layers of the TCP/IP stack, as shown in Figure 9.2. The most important of the internet- and transport-layer protocols are the building blocks for the application-layer protocols with which you interact more directly. These important internet- and transport-layer protocols include the following:

IP   The *Internet Protocol (IP)* is the core protocol in TCP/IP networking. Referring to Figure 9.2, IP is an internet-layer (aka a network-layer or layer 2) protocol. IP provides a "best effort" method for transferring packets between computers—that is, the packets aren't guaranteed to reach their destination. Packets may also arrive out of order or corrupted. Other components of the TCP/IP stack must deal with these issues, and have their own ways of doing so. IP is also the portion of TCP/IP with which IP addresses are associated. (The Real-World Scenario sidebar, "The Coming of IPv6," describes a change in the IP portion of TCP/IP that's underway.)

ICMP   The *Internet Control Message Protocol (ICMP)* is a simple protocol for communicating data. ICMP is most often used to send error messages between computers—for instance, to signal that a requested service isn't available. This is often done by modifying an IP packet and returning it to its sender, which means that ICMP is technically an internet-layer protocol, although it relies upon IP. In most cases, you won't use programs that generate ICMP packets on demand; they're created behind the scenes as you use other protocols. One exception is the `ping` program, which is described in more detail in "Testing Basic Connectivity."

UDP   The *User Datagram Protocol (UDP)* is the simplest of the common transport-layer (aka layer 3) TCP/IP protocols. It doesn't provide sophisticated procedures to correct for out-of-order packets, guarantee delivery, or otherwise improve the limitations of IP. This fact can be a problem, but it also means that UDP can be faster than more sophisticated tools that provide such improvements to IP. Common application-layer protocols that are built atop UDP include the Domain Name System (DNS), the Network File System (NFS), and many streaming media protocols.

TCP   The *Transmission Control Protocol (TCP)* may be the most widely-used transport-layer protocol in the TCP/IP stack. Unlike UDP, TCP creates full connections with error checking and correction as well as other features. These features simplify the creation of network protocols that must exchange large amounts of data, but the features come at a cost: TCP imposes a small performance penalty. Most of the application-layer protocols with which you may already be familiar, including the Simple Mail Transfer Protocol (SMTP), the Hypertext Transfer Protocol (HTTP), and the File Transfer Protocol (FTP), are built atop TCP.

You might notice that the name of the TCP/IP stack is built up of two of the TCP and IP protocol names. This is because these two protocols are so important for TCP/IP networking generally. TCP/IP, though, is much more than just these two protocols; it includes additional protocols, most of which (below the application layer) are rather obscure. On the other hand, a TCP/IP exchange need not use both TCP and IP—it could be a UDP or ICMP exchange, for instance.

Network Addressing

In order for one computer to communicate with another over a network, the computers need to have some way to refer to each other. The basic mechanism for doing this is provided by a network address, which can take several different forms, depending on the type of network hardware, protocol stack, and so on. Large and routed networks pose additional challenges to network addressing, and TCP/IP provides answers to these challenges. Finally, to address a specific program on a remote computer, TCP/IP uses a *port number,* which identifies a specific running program, something like the way a telephone extension number identifies an individual in a large company. The following sections describe all these methods of addressing.

The Coming of IPv6

Another alternative protocol stack is actually an extension of TCP/IP. The current version of the IP portion of TCP/IP is 4. A major upgrade to this is in the works, however, and it goes by the name *IPv6,* for IP version 6. IPv6 adds several features and improvements to TCP/IP, including standard support for more secure connections and support for many more addresses. Check `http://playground.sun.com/pub/ipng/html/ipng-main.html` or `http://www.ipv6forum.com` for detailed information on IPv6.

TCP/IP supports a theoretical maximum of about 4 billion addresses. Although this may sound like plenty, those addresses have not been allocated as efficiently as possible. Therefore, as the Internet has expanded, the number of truly available addresses has been shrinking at a rapid rate. IPv6 raises the number of addresses to 2^{128}, or 3.4×10^{38}. This is enough to give every square millimeter of land surface on Earth 2.2×10^{18} addresses.

IPv6 is starting to emerge as a real networking force in many parts of the world. The United States, though, is lagging behind on IPv6 deployment. The Linux kernel includes IPv6 support, so you can use it if you need to. Chances are that by the time the average office will need IPv6, it will be standard. Configuring a system for IPv6 is somewhat different from configuring it for IPv4, which is what this chapter describes.

Types of Network Addresses

Consider an Ethernet network. When an Ethernet frame leaves one computer, it is normally addressed to another Ethernet card. This addressing is done using low-level Ethernet features, independent of the protocol stack in question. Recall, however, that the Internet is composed of many different networks that use many different low-level hardware components. A user might have a dial-up telephone connection (through a serial port) but connect to one server that uses Ethernet and another that uses Token Ring. Each of these devices uses a different type of low-level network address. TCP/IP requires something more to integrate across different types of network hardware. In total, three types of addresses are important when you are trying to understand network addressing: network hardware addresses, numeric IP addresses, and text-based hostnames.

Hardware Addresses

One of the characteristics of dedicated network hardware such as Ethernet or Token Ring cards is that they have unique *hardware addresses*, also known as *Media Access Control (MAC) addresses*, programmed into them. In the case of Ethernet, these addresses are 6 bytes in length, and they're generally expressed as hexadecimal (base 16) numbers separated by colons. You can discover the hardware address for an Ethernet card by using the `ifconfig` command. Type **`ifconfig eth`*n*, where *n* is the number of the interface (0 for the first card, 1 for the second, and so on). You'll see several lines of output, including one like the following:

```
eth0      Link encap:Ethernet  HWaddr 00:A0:CC:24:BA:02
```

This line tells you that the device is an Ethernet card and that its hardware address is 00:A0:CC:24:BA:02. What use is this, though? Certain low-level network utilities and hardware use the hardware address. For instance, network switches use it to direct data packets. The switch learns that a particular address is connected to a particular wire, and so it sends data directed at that address *only* over the associated wire. The *Dynamic Host Configuration Protocol (DHCP),* which is described in the upcoming section "DHCP Configuration," is a means of automating the configuration of specific computers. It has an option that uses the hardware address to consistently assign the same IP address to a given computer. In addition, advanced network diagnostic tools are available that let you examine packets that come from or are directed to specific hardware addresses.

For the most part, though, you don't need to be aware of a computer's hardware address. You don't enter it in most utilities or programs. It's important for what it does in general.

IP Addresses

Earlier, I said that TCP/IP, at least in its IPv4 incarnation, supports about 4 billion addresses. This figure is based on the size of the *IP address* used in TCP/IP: 4 bytes (32 bits). Specifically, 2^{32} = 4,294,967,296. Not all of these addresses are usable; some are overhead associated with network definitions, and some are reserved.

The 4-byte IP address and 6-byte Ethernet address are mathematically unrelated. Instead, the TCP/IP stack converts between the two using the *Address Resolution Protocol (ARP).* This protocol enables a computer to send a *broadcast* query—a message that goes out to all the computers on the local network. This query asks the computer with a given IP address to identify itself. When a reply comes in, it includes the hardware address, so the TCP/IP stack can direct traffic for a given IP address to the target computer's hardware address.

> **NOTE** The procedure for computers that aren't on the local network is more complex. For such computers, a router must be involved. Local computers send packets destined to distant addresses to routers, which send the packets on to other routers or to their destination systems.

IP addresses are usually expressed as four base-10 numbers (0–255) separated by periods, as in 192.168.29.39. If your Linux system's protocol stack is already up and running, you can discover its IP address by using `ifconfig`, as described earlier. The output includes a line like the following, which identifies the IP address (`inet addr`):

```
inet addr:192.168.29.39  Bcast:192.168.29.255   Mask:255.255.255.0
```

Although not obvious from the IP address alone, this address is broken down into two components: a network address and a computer address. The network address identifies a block of IP addresses that are used by one physical network, and the computer address identifies one computer within that network. The reason for this breakdown is to make the job of routers easier—rather than record how to direct packets destined for each of the 4 billion IP addresses, routers can be programmed to direct traffic based on packets' network addresses, which is a much simpler job.

The *network mask* (also known as the *subnet mask* or *netmask*) is a number that identifies the portion of the IP address that's a network address and the part that's a computer address. It's helpful to think of this in binary (base 2) because the netmask uses binary 1 values to represent the network portion of an address and binary 0 values to represent the computer address. The network portion ordinarily leads the computer portion. Expressed in base 10, these addresses usually consist of 255 or 0 values, 255 being a network byte and 0 being a computer byte. If a byte is part network and part computer address, it will have some other value. Another way of expressing a netmask is as a single number representing the number of network bits in the address. This number usually follows the IP address and a slash. For instance, 192.168.29.39/24 is equivalent to 192.168.29.39 with a netmask of 255.255.255.0—the last number shows the network portion to be three solid 8-bit bytes, hence 24 bits. The longer notation showing all 4 bytes of the netmask is referred to as *dotted quad* notation.

On modern networks, netmasks are often described in *Classless Inter-Domain Routing (CIDR)* form. Such network masks can be broken at any bit boundary for any address. For instance, 192.168.1.7 could have a netmask of 255.255.0.0, 255.255.255.0, 255.255.255.128, or various other values. (Keeping each byte at 0 or 255 reduces the odds of human error causing problems but sometimes isn't practical, depending on the required or desired sizes of subnets.) Traditionally, though, networks have been broken into one of several classes, as summarized in Table 9.1. Classes A, B, and C are for general networking use. Class D addresses are reserved for *multicasting*—sending data to multiple computers simultaneously. Class E addresses are reserved for future use. There are a few special cases within most of these ranges. For instance, the 127.*x.y.z* addresses are reserved for use as *loopback* (aka *localhost*) devices—these addresses refer to the computer on which the address is entered. Addresses in which all the machine bits are set to 1 refer to the network block itself—they're used for broadcasts. The ultimate broadcast address is 255.255.255.255, which sends data to all computers on a network segment. (Routers normally block packets directed to this address. If they didn't, the Internet could easily be brought to its knees by a few people flooding the network with broadcast packets.)

TABLE 9.1   Network Classes and Private Network Ranges

Class	Address Range	Reserved Private Addresses
A	1.0.0.0–127.255.255.255	10.0.0.0–10.255.255.255
B	128.0.0.0–191.255.255.255	172.16.0.0–172.31.255.255
C	192.0.0.0–223.255.255.255	192.168.0.0–192.168.255.255
D	224.0.0.0–239.255.255.255	none
E	240.0.0.0–255.255.255.255	none

Within each of the three general-use network classes is a range of addresses reserved for private use. Most IP addresses must be assigned to individual computers by a suitable authority, lest two systems on the Internet both try to use a single address. The reserved private address spaces, however, can be used by anybody. (These address blocks are sometimes referred to as RFC1918 addresses, after the standards document—RFC1918—in which they're defined.) The caveat is that routers normally drop packets sent to these addresses, effectively isolating them from the Internet as a whole. The idea is that these addresses may be safely used by small private networks. Today, they're often used behind *Network Address Translation (NAT)* routers, which enable arbitrary numbers of computers to "hide" behind a single system. The NAT router substitutes its own IP address on outgoing packets and then directs the reply to the correct system. This is very handy if you want to connect more computers to the Internet than you have IP addresses.

> I generally use reserved private addresses for examples in this book. Unless otherwise specified, these examples work equally well on conventional assigned (non-private) IP addresses.

IP address classes were designed to simplify routing; however, as the Internet evolved, they became restrictive. Thus, today they serve mainly as a way to set default netmasks, such as 255.0.0.0 for Class A addresses or 255.255.255.0 for Class C addresses. Most configuration tools set these netmasks automatically, but you can override the settings, if necessary.

IP addresses and netmasks are extremely important for network configuration. If your network doesn't use DHCP or a similar protocol to assign IP addresses automatically, you must configure your system's IP address manually. A mistake in this configuration can cause a complete failure of networking or more subtle errors, such as an inability to communicate with just some computers.

> Non-TCP/IP stacks have their own addressing methods. NetBEUI uses machine names; it has no separate numeric addressing method. AppleTalk uses two 16-bit numbers. These addressing schemes are independent from IP addresses.

Broadcasts

Earlier, I mentioned broadcasts. This is a type of network transmission that's sent to all the computers on a local network, or occasionally all of the computers on a remote network. Under TCP/IP, a broadcast is done by specifying binary 1 values in all of the machine bits of the IP address. The network portion of the IP address may be set to the network's regular value, and this is required for directed broadcasts—that is, those that are sent to a remote network. (Many routers drop directed broadcasts, though.) In many cases, broadcasts are specified by the use of 255.255.255.255 as an IP address. Packets directed at this address are sent to all the machines on a local network.

Because the broadcast address for a network is determined by the IP address and netmask, you can convert between the broadcast address and netmask, given one of these and a computer's IP address. If the netmask happens to consist of whole-byte values (expressed as 0 or 255 in dotted quad notation), the conversion is easy: Replace the IP address components that have 0 values in the dotted quad netmask with 255 values to get the broadcast address. For instance, consider a computer with an IP address of 172.24.21.201 and a netmask of 255.255.0.0. The final two elements of the netmask have 0 values, so you swap in 255 values for these final two elements in the IP address to obtain a broadcast address of 172.24.255.255.

In the case of a CIDR address that has non-255 and non-0 values in the netmask, the situation is more complex because you must resort to binary (base 2) numbers. For instance, consider a computer with an IP address of 172.24.21.201 with a netmask of 255.255.128.0 (that is, 172.24.21.201/17). Expressed in binary, these numbers are:

```
10101100 00011000 00010101 11001001
11111111 11111111 10000000 00000000
```

To create the broadcast address, you must set the top (network address) values to 1 when the bottom (netmask) value is 0. In this case, the result is:

```
10101100 00011000 01111111 11111111
```

Converted back into base 10 notation, the resulting broadcast address is 172.24.127.255. Fortunately, you seldom need to perform such computations. When configuring a computer, you can enter the IP address and netmask and let the computer do the binary computations.

Hostnames

Computers work with numbers, so it's not surprising that TCP/IP uses numbers as computer addresses. People, though, work better with names. For this reason, TCP/IP includes a way to link names for computers (known as *hostnames*) to IP addresses. In fact, there are *several* ways to do this, some of which are described in the next section, "Resolving Hostnames."

As with IP addresses, hostnames are composed of two parts: *machine names* and *domain names*. The former refers to a specific computer and the latter to a collection of computers. Domain names are not equivalent to the network portion of an IP address, though; they're completely independent concepts. Domain names are registered for use by an individual or organization, which may assign machine names within the domain and link those machine names to any arbitrary IP address desired. Nonetheless, there is frequently some correspondence between domains and network addresses because an individual or organization that controls a domain is also likely to want a block of IP addresses for the computers in that domain.

Internet domains are structured hierarchically. At the top of the hierarchy are the top-level domains (TLDs), such as `.com`, `.edu`, and `.uk`. These TLD names appear at the *end* of an Internet address. Some correspond to nations (such as `.uk` and `.us`, for the United Kingdom and the United States, respectively), but others correspond to particular types of entities (such as `.com` and `.edu`, which stand for commercial and educational organizations, respectively). Within each TLD are various domains that identify specific organizations, such as `sybex.com` for Sybex or `loc.gov` for the Library of Congress. These organizations may optionally break their domains into *subdomains,* such as `cis.upenn.edu` for the Computer and Information Science

department at the University of Pennsylvania. Even subdomains may be further subdivided into their own subdomains; this structure can continue for many levels but usually doesn't. Domains and subdomains include specific computers, such as `www.sybex.com`, Sybex's web server.

When you configure your Linux computer, you may need to know its hostname. This will be assigned by your network administrator and will be a machine within your organization's domain. If your computer isn't part of an organizational network (say, if it's a system that doesn't connect to the Internet at all, or if it connects only via a dial-up account), you'll have to make up a hostname. Alternatively, you can register a domain name, even if you don't use it for running your own servers. Check `http://www.icann.org/registrars/accredited-list.html` for pointers to accredited domain registrars. Most registrars charge between $10 and $15 per year for domain registration. If your network uses DHCP, it may or may not assign your system a hostname automatically.

> If you make up a hostname, choose an invalid TLD, such as `.invalid`. This will guarantee that you don't accidentally give your computer a name that legitimately belongs to somebody else. Such a name conflict could prevent you from contacting that system, and it could cause other problems as well, such as misdirected e-mail.

Resolving Hostnames

The *Domain Name System (DNS)* is a distributed database of computers that convert between IP addresses and hostnames. Every domain must maintain at least two DNS servers that can either provide the names for every computer within the domain or redirect a DNS query to another DNS server that can better handle the request. Therefore, looking up a hostname involves querying a series of DNS servers, each of which redirects the search until the server that's responsible for the hostname is found. In practice, this process is hidden from you because most organizations maintain DNS servers that do all the dirty work of chatting with other DNS servers. You need only point your computer to your organization's DNS servers. This detail may be handled through DHCP, or it may be information you need to configure manually, as described later in the section "Configuring Linux for a Local Network."

Sometimes, you need to look up DNS information manually. You might do this if you know the IP address of a server through non-DNS means and suspect your DNS configuration is delivering the wrong address or to check whether a DNS server is working at all. Several programs can be helpful in performing such checks:

nslookup   This program performs DNS lookups (on individual computers by default) and returns the results. It also sports an interactive mode in which you can perform a series of queries. This program is officially deprecated, meaning that it's no longer being maintained and will eventually be dropped from its parent package (`bind-utils` or `bind-tools` on most distributions). Thus, you should get in the habit of using `host` or `dig` instead of `nslookup`.

host   This program serves as a replacement for the simpler uses of `nslookup`, but it lacks an interactive mode, and of course many details of its operation differ. In the simplest case, you can

type **host** *target.name*, where *target.name* is the hostname or IP address you want to look up. You can add various options that tweak its basic operation; consult host's man page for details.

dig This program performs more complex DNS lookups than host. Although you can use it to find the IP address for a single hostname (or a hostname for a single IP address), it's more flexible than host.

whois You can look up information on a domain as a whole with this command. For instance, typing **whois sybex.com** reveals who owns the sybex.com domain, who to contact in case of problems, and so on. You may want to use this command with -H, which omits the lengthy legal disclaimers that many domain registries insist on delivering along with whois information. Check the man page for whois for information on additional options.

Sometimes DNS is overkill. For instance, you might just need to resolve a handful of hostnames. This might be because you're configuring a small private network that's not connected to the Internet at large or because you want to set up a few names for local (or even remote) computers that aren't in the global DNS database. For such situations, /etc/hosts may be just what you need. This file holds mappings of IP addresses to hostnames, on a one-line-per-mapping basis. Each mapping includes at least one name, and sometimes more:

```
127.0.0.1     localhost
192.168.7.23  apollo.luna.edu   apollo
```

In this example, the name localhost is associated with the 127.0.0.1 address and the names apollo.luna.edu and apollo are tied to 192.168.7.23. The first of these linkages is standard; it should exist in any /etc/hosts file. The second linkage is an example that you can modify as you see fit. The first name is a full hostname, including the domain portion; subsequent names on the line are aliases—typically the hostname without its full domain specification.

Once you've set up an /etc/hosts file, you can refer to computers listed in the file by name, whether or not those names are recognized by the DNS servers the computer uses. One major drawback to /etc/hosts is that it's a purely local file; setting a mapping in one computer's /etc/hosts file only affects name lookups performed by that computer. Thus, to do good on an entire network, you must modify the /etc/hosts files on all of the computers on the network.

Linux normally performs lookups in /etc/hosts before it uses DNS. You can, however, modify this behavior by editing the /etc/nsswitch.conf file, and specifically the hosts line, which lists the order of the files and dns options, which stand for /etc/hosts and DNS, respectively. Very old programs that use libc4 or libc5 rather than glibc look to the /etc/host.conf file and its order line instead of nsswitch.conf. Change the order of the hosts and bind items in this file to match the order of the files and dns items in /etc/nsswitch.conf.

In addition to /etc/hosts, Linux supports a file called /etc/networks. It works much like /etc/hosts, but it applies to network addresses, and it reverses the order of the names and the IP address on each line:

```
loopback 127.0.0.0
mynet 192.168.7.0
```

This example sets up two linkages: the `loopback` name to the 127.0.0.0/8 network and `mynet` for the 192.168.7.0/24 network. It's seldom necessary to edit this file.

Network Ports

Contacting a specific computer is important, but one additional type of addressing is still left: The sender must have an address for a specific program on the remote system. For instance, suppose you're using a web browser. The web server computer may be running more servers than just a web server—it might also be running an e-mail server or an FTP server, to name just two of many possibilities. Another number beyond the IP address enables you to direct traffic to a specific program. This number is a network port number, and programs that accesses a TCP/IP network typically do so through one or more ports.

> **NOTE** Port numbers are features of the UDP and TCP protocols. Some protocols, such as ICMP, do not use port numbers.

When they start up, servers tie themselves to specific ports, which by convention are associated with specific server programs. For instance, port 25 is associated with e-mail servers, and port 80 is used by web servers. Thus, a client can direct its request to a specific port and expect to contact an appropriate server. The client's own port number isn't fixed; it's assigned by the OS. Because the client initiates a transfer, it can include its own port number in the connection request, so clients don't need fixed port numbers. Assigning client port numbers dynamically also enables one computer to easily run several instances of a single client because they won't compete for access to a single port.

Clients and Servers

One important distinction is the one between clients and servers. A *client* is a program that initiates a network connection to exchange data. A *server* listens for such connections and responds to them. For instance, a web browser, such as Mozilla or Opera, is a client program. You launch the program and direct it to a web page, which means that the web browser sends a request to the web server at the specified address. The web server sends back data in reply to the request. Clients can also send data, however, as when you enter information in a web form and click a Submit or Send button.

The terms *client* and *server* can also be applied to entire computers that operate mostly in one or the other role. Thus, a phrase such as *web server* is somewhat ambiguous—it can refer either to the web server program or to the computer that runs that program. When this distinction is important and unclear from context, I clarify it (for instance, by referring to "the web server program").

Fortunately, for basic functioning, you need to do nothing to configure ports on a Linux system. You may have to deal with this issue if you run unusual servers, though, because you may need to configure the system to link the servers to the correct ports. This can sometimes involve editing the /etc/services file, which maps port numbers to names, enabling you to use names in super server configurations and elsewhere. This file consists of lines that begin with a name and end with a port number, including the type of protocol it uses (TCP or UDP):

```
ssh     22/tcp      # SSH Remote Login Protocol
ssh     22/udp      # SSH Remote Login Protocol
telnet  23/tcp
smtp    25/tcp
```

> **NOTE** Chapter 7's Table 7.1 summarizes the standard uses of many important servers' ports.

Configuring Linux for a Local Network

Now that you know something about how networking functions, the question arises: How do you implement networking in Linux? Most Linux distributions provide you with the means to configure a network connection during system installation. Therefore, chances are good that networking already functions on your system. In case it doesn't, though, the following sections summarize what you must do to get the job done. Actual configuration can be done using either the automatic DHCP tool or static IP addresses. Linux's underlying network configuration mechanisms rely on startup scripts and their configuration files, but you may be able to use GUI tools to do the job instead.

Network Hardware Configuration

The most fundamental part of network configuration is getting the network hardware up and running. In most cases, this task is fairly automatic—most distributions ship with system startup scripts that auto-detect the network card and load the correct driver module. If you recompile your kernel, building the correct driver into the main kernel file will also ensure that it's loaded at system startup.

If your network hardware isn't correctly detected, though, subsequent configuration (as described in the upcoming sections, "DHCP Configuration" and "Static IP Address Configuration") won't work. To correct this problem, you must load your network hardware driver. You can do this with the modprobe command:

```
# modprobe tulip
```

You must know the name of your network hardware's kernel module, though (tulip in this example). Chapter 3, "Configuring Hardware," describes the task of hardware configuration and activation in more detail.

DHCP Configuration

One of the easiest ways to configure a computer to use a TCP/IP network is to use DHCP, which enables one computer on a network to manage the settings for many other computers. It works like this: When a computer running a DHCP client boots up, it sends a broadcast in search of a DHCP server. The server replies (using nothing but the client's hardware address) with the configuration information the client needs to enable it to communicate with other computers on the network—most importantly the client's IP address and netmask and the network's gateway and DNS server addresses. The DHCP server may also give the client a hostname. The client then configures itself with these parameters. The IP address is not assigned permanently; it's referred to as a *DHCP lease,* and if it's not renewed, the DHCP server may give the lease to another computer. Therefore, from time to time the client checks back with the DHCP server to renew its lease.

Three DHCP clients are in common use on Linux: `pump`, `dhclient`, and `dhcpcd` (not to be confused with the DHCP server, `dhcpd`). Some Linux distributions ship with just one of these, but others ship with two or even all three. All distributions have a default DHCP client, though—the one that's installed when you tell the system you want to use DHCP at system installation time. Those that ship with multiple DHCP clients typically enable you to swap out one for another simply by removing the old package and installing the new one.

Ideally, the DHCP client runs at system bootup. This is usually handled either by its own SysV startup file, as described in Chapter 6, "The Boot Process and Scripts," or as part of the main network configuration startup file (typically a SysV startup file called `network` or `networking`). The system often uses a line in a configuration file to determine whether to run a DHCP client. For instance, Red Hat Linux sets this option in a file called `/etc/sysconfig/network-scripts/ifcfg-eth0` (this filename may differ if you use something other than a single Ethernet interface). The line in question looks like this:

```
BOOTPROTO=dhcp
```

If the `BOOTPROTO` variable is set to something else, changing it as shown here will configure the system to use DHCP. It's usually easier to use a GUI configuration tool to set this option, however.

Once a DHCP client is configured to run when the system boots, the configuration task is done—at least, if everything works as it should. On very rare occasion, you may need to tweak DHCP settings to work around client/server incompatibilities or to have the DHCP client do something unusual. Consult the `man` page for your DHCP client if you need to make changes. You'll then have to modify its SysV startup script or a file to which it refers in order to change its operation.

Static IP Address Configuration

If a network lacks a DHCP server, you must provide basic network configuration options manually. You can set these options using interactive commands, as described shortly, but to set them in the long term, you adjust a configuration file such as `/etc/sysconfig/network-scripts/ifcfg-eth0`. Listing 9.1 shows a typical `ifcfg-eth0` file, configured to use a static IP address. (Note that this file's exact location and name may vary from one distribution to another.)

Listing 9.1: A Sample Network Configuration File

```
DEVICE=eth0
BOOTPROTO=static
IPADDR=192.168.29.39
NETMASK=255.255.255.0
NETWORK=192.168.29.0
BROADCAST=192.168.29.255
GATEWAY=192.168.29.1
ONBOOT=yes
```

Several specific items are required, or at least helpful, for static IP address configuration:

IP address   You can set the IP address manually via the `ifconfig` command (described in more detail shortly) or via the `IPADDR` item in the configuration file.

Network mask   The netmask can be set manually via the `ifconfig` command or via the `NETMASK` item in a configuration file.

Gateway address   You can manually set the gateway via the `route` command. To set it permanently, you need to adjust a configuration file, which may be the same configuration file that holds other options or another file, such as `/etc/sysconfig/network/routes`. In either case, the option is likely to be called `GATEWAY`. The gateway isn't necessary on a system that isn't connected to a wider network—that is, if the system works *only* on a local network that contains no routers.

DNS settings   In order for Linux to use DNS to translate between IP addresses and hostnames, you must specify at least one DNS server in the `/etc/resolv.conf` file. Precede the IP address of the DNS server by the keyword `nameserver`, as in `nameserver 192.168.29.1`. You can include up to three `nameserver` lines in this file. Adjusting this file is all you need to do to set the name server addresses; you don't have to do anything else to make the setting permanent.

The network configuration script may hold additional options, but most of these are related to others. For instance, Listing 9.1 has an option specifying the interface name (`DEVICE=eth0`), another that tells the computer to assign a static IP address (`BOOTPROTO=static`), and a third to bring up the interface when the computer boots (`ONBOOT=yes`). The `NETWORK` and `BROADCAST` items in Listing 9.1 are derived from the `IPADDR` and `NETMASK` items, but you can change them if you understand the consequences.

If you aren't sure what to enter for the basic networking values (the IP address, network mask, gateway address, and DNS server addresses), you should consult your network administrator. *Do not* enter random values or values you make up that are similar to those used by other systems on your network. Doing so is unlikely to work at all, and it could conceivably cause a great deal of trouble—say, if you mistakenly use an IP address that's reserved for another computer.

As just mentioned, the `ifconfig` program is critically important for setting both the IP address and netmask. This program can also display current settings. Basic use of `ifconfig` to bring up a network interface resembles the following:

```
ifconfig interface up addr netmask mask
```

For instance, the following command brings up eth0 (the first Ethernet card) using the address 192.168.29.39 and the netmask 255.255.255.0:

```
# ifconfig eth0 up 192.168.29.39 netmask 255.255.255.0
```

This command links the specified IP address to the card so that the computer will respond to the address and claim to be that address when sending data. It doesn't, though, set up a route for traffic beyond your current network. For that, you need to use the route command:

```
# route add default gw 192.168.29.1
```

Substitute your own gateway address for 192.168.29.1. (Routing and the route command are described in more detail shortly, in "Configuring Routing.") Both ifconfig and route can display information on the current network configuration. For ifconfig, omit up and everything that follows; for route, omit add and everything that follows. For instance, to view interface configuration, you might issue the following command:

```
# ifconfig eth0
eth0  Link encap:Ethernet  HWaddr 00:A0:CC:24:BA:02
      inet addr:192.168.29.39  Bcast:192.168.29.255   Mask:255.255.255.0
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:10469 errors:0 dropped:0 overruns:0 frame:0
      TX packets:8557 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:100
      RX bytes:1017326 (993.4 Kb)  TX bytes:1084384 (1.0 Mb)
      Interrupt:10 Base address:0xc800
```

When configured properly, ifconfig should show a hardware address (HWaddr), an IP address (inet addr), and additional statistics. There should be few or no errors, dropped packets, or overruns for both received (RX) and transmitted (TX) packets. Ideally, few (if any) collisions should occur, but some are unavoidable if your network uses a hub rather than a switch. If collisions total more than a few percent of the total transmitted and received packets, you may want to consider replacing a hub with a switch. To use route for diagnostic purposes, you might try the following:

```
# route
Kernel IP routing table
Destination Gateway        Genmask        Flags Metric Ref  Use Iface
192.168.29.0 *             255.255.255.0 U     0      0      0 eth0
127.0.0.0    *             255.0.0.0     U     0      0      0 lo
default      192.168.29.1 0.0.0.0        UG    0      0      0 eth0
```

This shows that data destined for 192.168.29.0 (that is, any computer with an IP address between 192.168.29.1 and 192.168.29.254) goes directly over eth0. The 127.0.0.0 network is a special interface that "loops back" to the originating computer. Linux uses this for some internal

networking purposes. The last line shows the *default route*, which describes what to do with everything that doesn't match any other entry in the routing table. This line specifies the default route's gateway system as 192.168.29.1. If it's missing or misconfigured, some or all traffic destined for external networks, such as the Internet, won't make it beyond your local network segment.

As with DHCP configuration, it's almost always easier to use a GUI configuration tool to set up static IP addresses, at least for new administrators. The exact locations of the configuration files differ from one distribution to another, so the examples listed earlier may not apply to your system.

Configuring Routing

As explained earlier, routers pass traffic from one network to another. You configure your Linux system to directly contact systems on the local network. You also give the computer a router's address, which your system uses as a gateway to the Internet at large. Any traffic that's not destined for the local network is directed at this router, which passes it on to its destination. In practice, there are likely to be a dozen or more routers between you and most Internet sites. Each router has at least two network interfaces and keeps a table of rules concerning where to send data based on the destination IP address. Your own Linux computer has such a table, but it's likely to be very simple compared to those on major Internet routers.

Linux can function as a router, which means it can link two or more networks together, directing traffic between them on the basis of its routing table. This task is handled, in part, by the route command. This command can be used to do much more than just specify a single gateway system, though, as was described earlier. A simplified version of the route syntax is as follows:

```
route {add | del} [-net | -host] target [netmask nm] [gateway gw]
➥[reject] [[dev] interface]
```

That is, you specify add or del along with a *target* (a computer or network address) and optionally other parameters. The -net and -host options force route to interpret the target as a network or computer address, respectively. The netmask option lets you set a netmask as you desire, and gateway lets you specify a router through which packets to the specified *target* should go. (Some versions of route use gw rather than gateway.) The reject keyword installs a blocking route, which refuses all traffic destined for the specified network. (This is *not* a firewall, though.) Finally, although route can usually figure out the interface device (for instance, eth0) on its own, you can force the issue with the dev option.

As an example, consider a network in which packets destined for the 172.20.0.0/16 subnet should be passed through the 172.21.1.1 router, which is not the default gateway system. You could set up this route with the following command:

```
# route add -net 172.20.0.0 netmask 255.255.0.0 gw 172.21.1.1
```

Incorrect routing tables can cause serious problems because some or all computers won't respond. You can examine your routing table by typing **route** alone and compare the results to what your routing table should be. (Consult a network administrator if you're not sure what your routing table should contain.) You can then delete incorrect routes and add new ones to replace them, if necessary. Ultimately, of course, changing your configuration files is the best solution, but typing a couple of `route` commands will do the trick in the short term.

One more thing you may need to do if you're setting up a router is enabling routing. Ordinarily, a Linux system will not forward packets it receives from one system that are directed at another system. If Linux is to act as a router, though, it must accept these packets and send them on to the destination network (or at least to an appropriate gateway). To enable this feature, you must modify a key file in the `/proc` filesystem:

```
# echo "1" > /proc/sys/net/ipv4/ip_forward
```

This command enables IP forwarding. Permanently setting this option requires modifying a configuration file. Some distributions set it in `/etc/sysctl.conf`:

```
net.ipv4.ip_forward = 1
```

Other distributions use other configuration files and options, such as `/etc/sysconfig/sysctl` and its `IP_FORWARD` line. If you can't find it, try using `grep` to search for `ip_forward` or `IP_FORWARD`, or enter the command to perform the change manually in a local startup script.

Using GUI Configuration Tools

Most distributions include their own GUI configuration tools for network interfaces. For instance, Fedora and Red Hat ship with a custom GUI tool called Network Configuration and a text-mode tool called `netconfig`, and SuSE has a text-mode and GUI tool called YaST. The details of operating these programs differ, but the GUI configuration tool provides a means to enter the information described earlier.

Although the LPI exam doesn't cover GUI network configuration tools, they're generally easier to locate and use than the configuration files in which settings are stored. Thus, you may want to look for your distribution's tool and learn to use it. Once you understand the principles of network configuration (IP addresses, DHCP, and so on), you shouldn't have trouble entering the necessary information in the GUI fields.

The precise details of how to configure a Linux system using GUI tools differ from one distribution to another. For instance, SuSE's YaST doesn't lay out its options in precisely the same way as Fedora's Network Configuration tool. The basic principles are the same, though; you must choose whether to use static IP address assignment or an automatic system such as DHCP and enter a number of key options, depending on what configuration method you choose.

Hostname Configuration

The hostnames described earlier (in "Hostnames") are configured in a couple of ways:

On DNS   Your network administrator should be able to add an entry for your system to your network's DNS server. This entry should make your computer addressable by name by other computers on your local network, and perhaps on the Internet at large. Alternatively, remote systems' `/etc/hosts` files can be modified to include your system. Chapter 10 provides some basics on DNS server configuration.

On your local computer   Various local programs should know your computer's name. For instance, you might want to have your hostname displayed as part of a command prompt or entered automatically in e-mail programs. For this task, you must set your hostname locally. Note that this is entirely independent of your DNS hostname. In theory, you can set the two to very different values, but this practice is likely to lead to confusion and perhaps even failure of some programs to operate properly.

The most basic tool for setting your hostname locally is called, appropriately enough, `hostname`. Type the command alone to see what your hostname is, or type it with a new name to set the system's hostname to that name:

```
# hostname nessus.example.com
```

Similar commands, `domainname` and `dnsdomainname`, display or set the computer's domain name (such as `example.com`). The `domainname` command sets the domain name as used by Network Information System (NIS), whereas `dnsdomainname` sets the domain name as used by DNS. These commands don't affect remote servers, though, just the name given to programs that use calls designed for these servers.

Many Linux distributions look in the `/etc/hostname` or `/etc/HOSTNAME` file for a hostname to set at boot time. Thus, if you want to set your hostname permanently, you should look for these files, and if one is present, you should edit it. If your system has neither file, consult its documentation; it's conceivable that your distribution stores its hostname in some unusual location.

EXERCISE 9.1

Configuring a Network Connection

In this exercise, you'll familiarize yourself with some of the tools used to configure basic network settings. You'll use these tools both to study and to change your network configuration. The assumption going into this exercise is that the system is correctly configured to use an Ethernet network, including both local network access and access to a larger network (probably the Internet) via a router.

Some of the procedures in this exercise can easily break your network connectivity if something goes wrong. If this happens, rebooting the computer is the simplest, albeit a radical, way to recover.

To study and modify your system's network configuration, follow these steps:

1. Log into the Linux system as a normal user.

2. Launch an xterm from the desktop environment's menu system, if you used a GUI login method.

3. Acquire root privileges. You can do this by typing **su** in an xterm, by selecting Session ➢ New Root Console from a Konsole, or by using sudo (if it's configured) to run the commands in the following steps.

4. Type **ifconfig**. This command displays information on your local network settings for all your network interfaces. Most systems will have both a loopback interface (lo) and an Ethernet interface (eth0). Look for a line in the Ethernet section that includes the string inet addr:. The following 4-byte number is your IP address. Write it down, as well as the value of your netmask (Mask:). Study the other information in this output, too, such as the number of received (RX) and transmitted (TX) packets, the number of errors, the number of collisions, and the Ethernet adapter's hardware address.

5. Type **route -n**. The output is your computer's routing table information. This normally includes information on the loopback network address (127.0.0.0/24), the local network address, and a default route (identified as the route for 0.0.0.0). Some systems may display fewer or additional lines, depending on local configuration. The default route includes an IP address under the Gateway column. Write that address down.

6. Use ping to test connectivity to both local and remote computers. (This command is described in more detail shortly, in "Testing Basic Connectivity.") You'll need the name or IP address of at least one local computer and at least one distant computer (beyond your local router). Type **ping *address***, where *address* is the name or IP address of each test machine. Perform this test for localhost or 127.0.0.1, your own machine (use the IP address you noted in step 4), your local router (use the IP address you noted in step 5), and a distant computer (if you're connected to the Internet, you can use an Internet-accessible site, such as www.linux.org). All of these ping tests should be successful. Note, however, that some systems are configured to ignore packets sent by ping. Thus, some of these tests may fail if you run into such systems. You can learn the configuration of local systems from their administrators, but for Internet sites, you might want to simply try another site if the first one you test fails.

7. Bring down the local Ethernet connection by typing **ifconfig eth0 down**.

8. Repeat steps 4–6. Note that the eth0 interface is no longer shown when you type ifconfig, all routes associated with it have been removed from the routing table, and pinging systems accessible from the interface will no longer work. (Linux retains some information about its former Ethernet link, though, so you may still be able to ping the computer itself via its former eth0 address.)

9. Bring the local Ethernet connection back up by typing **ifconfig eth0 up *address* netmask *mask***, where *address* is the original IP address and *mask* is the original netmask, both as identified in step 4.

10. Repeat steps 4–6. Note that the ifconfig command automatically added back your local network to the routing table but that the default route is still missing. As a result, you can't contact any systems that are located off the local network. If your DNS server is such a system, this means your ability to contact even local machines by name may be impaired as well.

11. Restore the default route by typing **route add default gw *gateway***, where *gateway* is the router address you identified in step 5.

12. Repeat steps 4–6. If your network configuration is typical, all connectivity should be restored. (Some more exotic systems may still be lacking certain routes, though.)

Configuring Linux as a PPP Client

A conventional telephone modem is the low end of network connectivity. This device turns the public telephone system into a computer networking medium, linking precisely two points together. In its simplest form, a modem can be used to initiate a text-mode connection using a *terminal program*—a program that enables remote text-based logins but nothing else. Today, though, a modem is more often used in conjunction with PPP. PPP establishes a TCP/IP link between the two computers, so you can use any of the many TCP/IP-based tools. Most PPP accounts, though, are designed to be used for brief periods at a time, not continuously. Modem connections are also much slower than most other types of network connection. Therefore, running servers on PPP-connected systems is usually inadvisable because most servers require always-on Internet connections, and many need more speed than a PPP link can provide. Some Internet service providers (ISPs) do offer full-time PPP links, though.

To initiate a PPP connection, you must have PPP software installed on your Linux system. The most important PPP package is known as pppd, for "PPP daemon." This utility can both initiate PPP links and respond to attempts to initiate them. The following sections describe the former. In addition to the basic PPP utilities, another tool, wvdial, adds extra features to PPP.

Making Basic PPP Connections

In Linux, a PPP connection usually requires an entry in a file called /etc/ppp/pap-secrets or /etc/ppp/chap-secrets. Both files use the same format. They provide information that's passed between the PPP client and server for authentication using the Password Authentication Protocol (PAP) or Challenge-Handshake Authentication Protocol (CHAP). Because PPP was

designed for use over public dial-up telephone lines, the caller must normally present a user-name and password to the other system; PAP and CHAP are merely protocols for doing this in a standard way. The format of lines in the secrets files is as follows:

```
username  server  password  IP_address
```

The *username* and *password* values are your username and password, respectively, on the remote PPP system. Enter the values obtained from your ISP. The *server* value is the name of the system to which you're connecting. Normally, it's an asterisk (*), signifying that pppd will connect to any computer. *IP_address* is the IP address that pppd expects to get. This will normally be blank, meaning that the system will accept any IP address.

> **WARNING**   The PAP/CHAP secrets file contains a password in clear text, which is potentially dangerous. Ensure that the file is readable only to those users who must be able to read it—possibly only root, or perhaps root and whoever must be able to initiate PPP connections. Don't use your ISP's PPP password for any other purpose, such as for authenticating yourself on your Linux system or on any other system.

Connecting from the command line requires modifying certain connection scripts. These are called ppp-on, ppp-on-dialer, and ppp-off. The first two start a connection, and the third breaks it. These scripts are often stored in a documentation directory, such as /usr/share/doc/ppp-2.4.2/scripts. Copy them to a convenient binary directory that's on your path, such as /usr/local/bin. You must then modify them with information relevant to your ISP:

- In ppp-on, locate the lines that begin TELEPHONE=, ACCOUNT=, and PASSWORD=, and modify them so that they're appropriate for your ISP and account. (The ACCOUNT and PASSWORD variables should contain dummy values if you use PAP or CHAP, as is almost always the case.)

- Check that the DIALER_SCRIPT variable in ppp-on points to the correct location of ppp-on-dialer. The default location is /etc/ppp.

- Check the call to pppd in the last lines of ppp-on. Most of the parameters to this call are quite cryptic, but you should at least be able to confirm that it's using the correct modem device filename and speed. RS-232 serial modems generally use /dev/ttyS0 or /dev/ttyS1 as the filename. 115200 is an appropriate speed in most cases, but the default is 38400.

- Check the ppp-on-dialer script. This script includes a "chat" sequence—a series of strings the program expects to see from the modem or remote system in one column and a series of responses in another column. Listing 9.2 shows a typical chat sequence from a ppp-on-dialer script. You may need to log on using a terminal program like Seyon or minicom and then capture to disk the prompts your ISP uses to ask for your username and password; you'll then need to modify the last two lines of the script in order to make it work. Alternatively, you may have to comment out the last two lines by preceding them with hash marks (#) and remove the backslash (\) from the CONNECT line if your ISP uses PAP or CHAP.

> The chat program expects a single line; its input is formatted in columns in ppp-on-dialer only for the convenience of humans. The backslashes ending most lines signify line continuations so that chat interprets multiple input lines as a single line. Only the final line should lack a backslash.

Listing 9.2: A Typical Chat Sequence

```
exec chat -v                                         \
        TIMEOUT       3                              \
        ABORT         '\nBUSY\r'                     \
        ABORT         '\nNO ANSWER\r'                \
        ABORT         '\nRINGING\r\n\r\nRINGING\r'   \
        ''            \rAT                           \
        'OK-+++\c-OK' ATHO                           \
        TIMEOUT       30                             \
        OK            ATDT$TELEPHONE                 \
        CONNECT       ''                             \
        ogin:--ogin:  $ACCOUNT                       \
        assword:      $PASSWORD
```

Although you can set many pppd options in the last lines of the ppp-on script, as just described, another approach is to edit the /etc/ppp/options file. This file contains pppd options, one per line.

When you're done making these changes, type **ppp-on** (preceding it with a complete path, if necessary) as root to test the connection. If all goes well, your system should dial the modem, link up, and give you Internet access. If this fails to occur, check the last few lines of /var/log/messages with a command such as **tail -n 20 /var/log/messages**. You should see some sort of error messages, which may help you to diagnose the problem. To terminate a connection, type **ppp-off**.

> The /etc/ppp/ip-up and /etc/ppp/ip-down scripts referred to in the LPIC objectives are executed automatically by pppd behind the scenes. You shouldn't need to edit these files or call them directly.

Using Supplemental PPP Tools

The pppd daemon, as well as the ppp-on and related scripts, are adequate tools for initiating and managing a PPP connection—but they're little more than adequate. Configuring a system to connect using pppd and its regular scripts is tedious because you must edit these scripts and because a wide variety of problems can crop up. These problems can be tricky to debug, making the process of configuring PPP a frustrating one.

One possible solution to this problem is wvdial. This program serves as a text-mode front end to pppd. You can think of it as a ppp-on script with some degree of intelligence. When run, wvdial checks its configuration files (described shortly), dials your ISP, and automatically figures out how to initiate a PPP connection. For instance, if your ISP prompts for a username and password, wvdial sends them automatically; but if your ISP uses PAP or CHAP, wvdial detects this fact and exchanges the password using the appropriate protocol. In most cases, therefore, wvdial is simpler to configure and use than the traditional ppp-on and related tools.

Of course, wvdial does require *some* configuration. You must give it a telephone number, your username, and your password, for instance. You can also set various other options. These details are handled in the global /etc/wvdial.conf file and in individual users' ~/.wvdialrc files. Both files have the same format, which consists of section names in square brackets ([]) followed by lines containing option assignments that use equal signs (=). The most important section is Dialer Defaults, which sets the defaults that are to be used if wvdial is called without any other options:

```
[Dialer Defaults]
Modem = /dev/ttyS1
Baud = 57600
Init = ATZ
Init2 = AT S11=50
Phone = 555-4425
Username = agbell
Password = yi3Wt#TD
```

This configuration sets options using names that are easy to interpret—Modem is the modem's device filename, Baud is the RS-232 serial port's speed, and so on. The Init and Init2 strings set the initialization strings that are sent to the modem prior to dialing. In most cases, setting Init = ATZ and omitting the Init2 string is sufficient. Some modems work better with additional options, though, as shown in the Init2 string in this example. Although it's not specified explicitly, wvdial also automatically reconnects if your connection drops unexpectedly. (You can change this behavior by setting the Auto Reconnect = off option.)

> **WARNING**   Because the wvdial configuration file usually contains a password, you should protect it very carefully, just as you'd protect the PAP or CHAP secrets file.

Additional wvdial configuration sections typically take the name Dialer *ID*, where *ID* is an arbitrary name, such as Modem2 or RoamingISP. You can then change any number of options from those set in the Dialer Defaults section. For instance, you can set a new phone number, username, and password if you have accounts with two ISPs.

To use wvdial, you normally just type the program's name: **wvdial**. If you've configured additional Dialer sections, you can add the section name, as in **wvdial Modem2** to use the Modem2 configuration section.

Several other tools are available to help with PPP configuration and use. Most notable are GUI tools, such as KPPP, which is part of the K Desktop Environment (KDE). These programs enable you to manage a PPP session using a point-and-click interface that's similar to the one provided by Windows. In fact, a GUI front end to `wvdial` is available: QtWvDialer (`http://www.mtoussaint.de/qtwvdialer.html`).

⊕ Real World Scenario

Using PPP with ISDN or DSL

PPP has traditionally been used over RS-232 serial ports, usually in conjunction with a modem. PPP has been adapted to several other uses, though. Two of these are *Integrated Services Digital Network (ISDN)* and *Digital Subscriber Line (DSL)* connections. DSL is available in several variants, such as Symmetric DSL (SDSL) and Asymmetric DSL (ADSL).

Using PPP with ISDN works much like using it with an RS-232 serial port. The main difference is that you must change the device identifier to that of your ISDN port. This port might be an RS-232 serial port, but it's more likely to be a specialized port provided by your ISDN adapter. Consult the driver documentation for your ISDN adapter for details.

Using PPP with DSL is a bit different. Most commonly, *PPP over Ethernet (PPPoE)* is used. The Roaring Penguin PPPoE client (RP-PPPoE; `http://www.roaringpenguin.com/penguin/open_source_rp-pppoe.php`) is a common PPPoE client for Linux. This software works best with external Ethernet-based DSL modems rather than internal or USB-interfaced modems. It includes a configuration script that collects necessary information and creates dialing scripts that you can call manually or run as part of your system startup process.

An alternative to using RP-PPPoE is to purchase an external broadband router. These devices manage the PPPoE connection to your ISP on one interface and present ordinary TCP/IP Ethernet, including their own DHCP servers, on the other interface. The result is that you can configure your system just as you would for an ordinary Ethernet network with a DHCP server. These routers enable you to link multiple systems to a single DSL connection. They also work with cable modem systems (which typically use DHCP and normal Ethernet configurations). Broadband routers have security advantages because they block all incoming connection attempts—they're NAT routers, as described earlier, in "IP Addresses."

Diagnosing Local and PPP Connections

Network configuration is a complex topic, and unfortunately, things don't always work as planned. Fortunately, there are a few commands you can use to help diagnose a problem. Four of these are `ping`, `traceroute`, `netstat`, and `tcpdump`. Each of these commands exercises the

network in a particular way and provides information that can help you track down the source of a problem. You can also use some common network programs that aren't primarily debugging tools in your debugging efforts.

Testing Basic Connectivity

The most basic network test is the `ping` command, which sends a simple ICMP packet to the system you name (via IP address or hostname) and waits for a reply. In Linux, `ping` continues sending packets once every second or so until you interrupt it with a Ctrl+C keystroke. (You can instead specify a limited number of tests via the `-c` *num* option.) Here's an example of its output:

```
$ ping -c 4 speaker
PING speaker (192.168.1.1) 56(84) bytes of data.
64 bytes from speaker.example.com (192.168.1.1): icmp_seq=1 ttl=64 time=0.194ms
64 bytes from speaker.example.com (192.168.1.1): icmp_seq=2 ttl=64 time=0.203ms
64 bytes from speaker.example.com (192.168.1.1): icmp_seq=3 ttl=64 time=0.229ms
64 bytes from speaker.example.com (192.168.1.1): icmp_seq=4 ttl=64 time=0.217ms


--- speaker ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 0.194/0.210/0.229/0.022 ms
```

This command sent four packets and waited for their return, which occurred quite quickly (in an average of 0.210ms) because the target system was on the local network. By pinging systems on both local and remote networks, you can isolate where a network problem occurs. For instance, if you can ping local systems but not remote systems, the problem is most probably in your router configuration. If you can ping by IP address but not by name, the problem is with your DNS configuration.

Tracing a Route

A step up from `ping` is the `traceroute` command, which sends a series of three test packets to each computer between your system and a specified target system. The result looks something like this:

```
$ traceroute -n 10.1.0.43
traceroute to 10.1.0.43 (10.1.0.43), 30 hops max, 52 byte packets
 1  192.168.1.254  1.021 ms  36.519 ms  0.971 ms
 2  10.10.88.1  17.250 ms  9.959 ms  9.637 ms
 3  10.9.8.173  8.799 ms  19.501 ms  10.884 ms
 4  10.9.8.133  21.059 ms  9.231 ms  103.068 ms
 5  10.9.14.9  8.554 ms  12.982 ms  10.029 ms
 6  10.1.0.44  10.273 ms  9.987 ms  11.215 ms
 7  10.1.0.43  16.360 ms  *  8.102 ms
```

The -n option to this command tells it to display target computers' IP addresses rather than their hostnames. This can speed up the process a bit, particularly if you're having DNS problems, and it can sometimes make the output easier to read—but you might want to know the hostnames of problem systems because that can help you pinpoint who's responsible for a problem.

This sample output shows a great deal of variability in response times. The first hop, to 192.168.1.254, is purely local; this router responded in 1.021, 36.519, and 0.971 milliseconds (ms) to its three probes. (Presumably the second probe caught the system while it was busy with something else.) Probes of most subsequent systems are in the 8–20ms range, although one is at 103.068ms. The final system only has two times; the middle probe never returned, as the asterisk (*) on this line indicates.

Using traceroute, you can localize problems in network connectivity. Highly variable times and missing times can indicate a router that's overloaded or that has an unreliable link to the previous system on the list. If you see a dramatic jump in times, it typically means that the physical distance between two routers is great. This is common in intercontinental links. Such jumps don't necessarily signify a problem, though, unless the two systems are close enough that a huge jump isn't expected.

What can you do with the traceroute output? Most immediately, traceroute is helpful in determining whether a problem in network connectivity exists in a network for which you're responsible. For instance, the variability in the first hop of the preceding example could indicate a problem on the local network, but the lost packet associated with the final destination most likely is not a local problem. If the trouble link is within your jurisdiction, you can check the status of the problem system, nearby systems, and the network segment in general.

Checking Network Status

Another useful diagnostic tool is netstat. This is something of a Swiss Army knife of network tools because it can be used in place of several others, depending on the parameters it is passed. It can also return information that's not easily obtained in other ways. Some examples include the following:

Interface information   Pass netstat the --interface or -i parameter to obtain information on your network interfaces similar to what ifconfig returns. (Some versions of netstat return information in the same format, but others display the information differently.)

Routing information   You can use the --route or -r parameter to obtain a routing table listing similar to what the route command displays.

Masquerade information   Pass netstat the --masquerade or -M parameter to obtain information on connections mediated by Linux's NAT features, which often go by the name "IP masquerading." NAT enables a Linux router to "hide" a network behind a single IP address. This can be a good way to stretch limited IP addresses.

Program use   Some versions of netstat support the --program or -p parameters, which attempt to provide information on the programs that are using network connections. This attempt isn't always successful, but it often is, so you can see what programs are making outside connections.

Open ports   When used with various other parameters, or without any parameters at all, `netstat` returns information on open ports and the systems to which they connect.

Keep in mind that `netstat` is a very powerful tool, and its options and output aren't entirely consistent from one distribution to another. You may want to peruse its `man` page and experiment with it to learn what it can do.

Examining Raw Network Traffic

One advanced network troubleshooting tool is `tcpdump`. This program is a *packet sniffer*, which is a program that can intercept network packets and log them or display them on the screen. Packet sniffers can be useful diagnostic tools because they enable you to verify that a computer is actually receiving data from other computers. They also enable you to examine the data in its raw form, which can be useful if you understand enough of the protocol's implementation details to spot problems.

> **WARNING**   Although packet sniffers are useful diagnostic tools, they can also be abused. For instance, unscrupulous individuals can run packet sniffers to capture passwords that others send over the network. Depending on your network configuration, this trick can work even if the packet sniffer isn't running on either the sending or the receiving computer. For this reason, many organizations have policies forbidding the use of packet sniffers except under limited circumstances. Thus, before running a packet sniffer, you should obtain written permission to use such a program from an individual who is authorized to grant such permission. Failure to do so could lead you into serious trouble, possibly up to losing your job or even being sued.

In its most basic form, you can use `tcpdump` by typing its name:

```
# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
19:31:55.503759 IP speaker.example.com.631 > 192.168.1.255.631: UDP,
➥length: 139
19:31:55.505400 IP nessus.example.com.33513 > speaker.example.com.domain:
  ➥46276+ PTR? 255.1.168.192.in-addr.arpa. (44)
19:31:55.506086 IP speaker.example.com.domain > nessus.example.com.33513:
  ➥46276 NXDomain* 0/1/0 (110)
```

The first thing to note about this command is that you must run it as `root`; ordinary users aren't allowed to monitor network traffic in this way. Once it's run, `tcpdump` summarizes what it's doing and then begins printing lines, one line for each packet it monitors. (Some of these lines can be quite long and so may take more than one line on your display.) These lines include a time stamp, a stack identifier (`IP` in all of these examples), the origin system name or IP address and port, the destination

system name or IP address and port, and packet-specific information. Ordinarily, `tcpdump` keeps displaying packets indefinitely, so you must terminate it by pressing Ctrl+C. Alternatively, you can pass it the `-c` *num* option to have it display *num* packets and then quit.

Even this basic output can be very helpful. For instance, consider the preceding example of three packets, which was captured on `nessus.example.com`. This computer successfully received one broadcast packet (addressed to 192.168.1.255) from `speaker.example.com`'s UDP port 631, sent a packet to `speaker.example.com`, and received a packet from that system directed at `nessus.example.com` rather than sent as a broadcast. This sequence verifies that at least minimal communication exists between these two computers. If you were having problems establishing a connection, you could rule out a whole range of possibilities based on this evidence, such as faulty cables or a firewall that's blocking traffic.

If you need more information, `tcpdump` provides several options that enhance or modify its output. These include `-A` to display packet contents in ASCII, `-D` to display a list of interfaces to which `tcpdump` can listen, `-n` to display all addresses numerically, `-v` (and additional `-v` options, up to `-vvv`) to display additional packet information, and `-w` *file* to write the capture packets to the specified *file*. Consult `tcpdump`'s `man` page for more details on these options and for additional options.

Using Additional Tools

In addition to specialized network diagnostic programs, you can use some common user programs as debugging tools. One of the most useful of these may be Telnet. This program and protocol is mainly a remote login tool; type the program name followed by the name of a remote system to receive a login prompt on that system:

```
$ telnet speaker
Trying 192.168.1.1...
Connected to speaker.
Escape character is '^]'.

speaker login: harry
Password:
Last login: Mon Apr 25 21:48:44 from nessus.example.com
Have a lot of fun...
harry@speaker:~>
```

> **WARNING** Telnet is a poor choice as a remote login protocol because it's entirely unencrypted. As a general rule, you should remove the Telnet server from your system's configuration (it's usually launched from a super server) and never use the `telnet` client program. It can be a useful lowest-common-denominator protocol on sufficiently protected private networks, though, and it can also be a handy tool for debugging, as described next. Chapter 10 describes the Secure Shell (SSH), which is a much safer alternative to Telnet.

You can also use Telnet to debug network protocols; if you give it a port number after the remote hostname, the `telnet` program connects to that port, enabling you to interact with the server:

```
$ telnet speaker 25
Trying 192.168.1.1...
Connected to speaker.
Escape character is '^]'.
220 speaker.example.com ESMTP Postfix
HELO nessus.example.com
250 speaker.example.com
```

This example connects to port 25, which is used by mail servers. After connecting, I entered a HELO command, which is used by the Simple Mail Transfer Protocol (SMTP) to identify a client, and the remote system responded with a 250 code, which indicates an accepted command.

Of course, to use Telnet in this way, you must know a great deal about the protocol. Even without this knowledge, though, you can use Telnet to test whether or not a server is running: If you try to connect but get a `Connection refused` error message, you know that a remote server is not running or is inaccessible for some reason (say, because it's being blocked by a firewall). If you get in (to the `Escape character` message shown in the earlier examples or beyond), the server is running, although it might not be working correctly. This test will only work for protocols that use TCP, though. Some simple tools use UDP instead, and Telnet won't connect with them.

Sometimes the File Transfer Protocol (FTP) can be a useful diagnostic tool, as well. This program, as its name suggests, enables you to transfer files between systems. To use it, type the program name followed by the FTP server's name. You'll then see a login prompt and be able to issue FTP commands:

```
$ ftp speaker
Connected to speaker.
220 (vsFTPd 1.2.1)
Name (speaker:harry): harry
530 Please login with USER and PASS.
SSL not available
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> get zathras.wav
local: zathras.wav remote: zathras.wav
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for zathras.wav (109986 bytes).
```

```
226 File send OK.
109986 bytes received in 0.104 secs (1e+03 Kbytes/sec)
ftp> quit
221 Goodbye.
```

This example retrieves a single file, `zathras.wav`, from the remote computer. In addition to `get`, which retrieves files, you can issue commands such as `put`, to upload a file; `ls` or `dir`, to display the remote system's directory contents; `cd`, to change directories on the remote system; `delete`, to remove a file; and `quit` or `exit`, to exit from the program. You can use the `help` or `?` command to see a list of available `ftp` commands.

As with Telnet, FTP is a poor choice of protocol for security reasons. The same SSH protocol that can substitute for Telnet can also handle most FTP duties. One important exception exists to the rule not to use FTP, though: Anonymous FTP sites are a common tool for distributing public files on the Internet. You can download Linux itself from anonymous FTP sites. These sites typically take a username of `anonymous` and any password (your e-mail address is the conventional reply) and give you read access to their contents. In most cases, you can't upload files to anonymous FTP sites, though, and you can only access a limited number of files.

Using a Super Server

A *super server* is an unusual type of server. Instead of handling one network protocol itself, a super server functions as an intermediary for other servers, as described next in "The Role of a Super Server." The end result is increased flexibility, but not all servers need or work well with super servers. In Linux, two super servers are in common use: `inetd` and `xinetd`. Most systems run only one of these super servers, which play similar roles but do things differently from one another.

The Role of a Super Server

Super servers sit between calling client systems and some or all of the individual server programs they're attempting to access, as shown in Figure 9.3. As illustrated by this figure, though, super servers often do not manage *all* of the individual server programs run on a particular computer—some server programs don't work well via a super server and so manage their own connections.

A single super server can manage connections from multiple client computers, just as most server programs can. A single client computer can connect to multiple servers on a single server system, with or without a super server intermediary. From the point of view of the client computer, the super server effectively does not exist; it looks just like the target server program that it manages. In fact, the super server does minimal direct "talking" to the client; it listens for the initial connection and, once that connection has been detected and passes certain preliminary tests, passes it on to the target server program.

FIGURE 9.3   Super servers manage connections between clients and individual server programs.
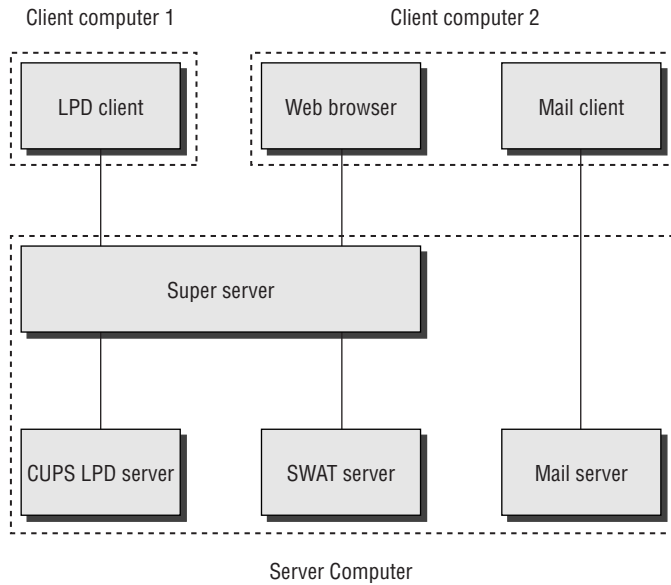
What, then, does a super server do? That is, why use one? Super servers offer several advantages over letting target servers listen for their connections directly:

Reduced overhead   Every program that a computer runs consumes resources, such as memory and CPU time. By running a super server, that overhead can be minimized, particularly for memory—a single super server can stand in for several target servers. This advantage is greatest for seldom-used servers, though; if a server is used frequently, it will be running most of the time whether or not a super server mediates access.

Unified configuration   Super servers can help simplify configuration by providing a single control point for multiple servers. You can go through the super server configuration file to enable or disable all the servers it manages. This advantage is limited, though, because many servers do *not* use super servers. That is, you can't assume that a server isn't running just because it's not listed in the super server configuration file.

Access control   Super servers can provide unified security features. For instance, they can restrict access to the servers based on time of day, calling IP address, and so on. The `inetd` super server does this with the help of another package, TCP Wrappers; `xinetd` implements these controls itself.

Of course, super servers aren't without their drawbacks. The most important of these is that servers launched via super servers typically respond slightly more slowly than do servers that run directly. The cause is simple: The super server must launch the server program every time it's accessed, and this process takes time. This effect is greatest for large servers; it can be trivial for

small servers. Another problem with super servers is that they can't manage every server program; some have requirements that super servers can't handle. For instance, a server might need to maintain information in memory between accesses, and if the super server launches a new instance for every access, this maintenance won't work.

In practice, you'll need to consult a server program's documentation to learn whether to launch it directly or via a super server. Some programs can be launched in either way, but most work best in one way or another. Most Linux distributions provide server packages with appropriate startup scripts to enable a server to launch in the correct way, although you may need to edit these scripts, particularly for servers that are handled by super servers.

Configuring *inetd*

Linux distributions have been slowly shifting from `inetd` to `xinetd`. Nonetheless, you may still find `inetd` in use on some systems. Type **ps ax | grep inetd** to see which super server is running on your system—the output should include a line with either the `inetd` or the `xinetd` command. Some systems run neither super server, though.

You control servers that launch via `inetd` through the `/etc/inetd.conf` file. This file consists of a series of lines, one for each server. A typical line resembles the following:

```
ftp stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.ftpd -l
```

> **NOTE** This and several subsequent examples refer to `in.ftpd`, an FTP server that was once quite popular but that's being replaced on many systems by other FTP servers. Some of these servers cannot be run from a super server, so using another server might not work in all of these cases.

Each line consists of several fields separated by one or more spaces. The meanings of these fields are as follows:

Service name   The first field (`ftp` in the preceding example) is the name of the service as it appears in the `/etc/services` file.

Socket type   The socket type entry tells the system what type of connection to expect—a reliable two-way connection (`stream`), a less reliable connection with less overhead (`dgram`), a low-level connection to the network (`raw`), or various others. The differences between these types are highly technical; your main concern in editing this entry should be to correctly type the value specified by the server's documentation.

Protocol   This is the TCP/IP transport-layer protocol used, usually `tcp` or `udp`.

Wait/no wait   For `dgram` socket types, this entry specifies whether the server connects to its client and frees the socket (`nowait`) or processes all its packets and then times out (`wait`). Servers that use other socket types should specify `nowait` in this field.

User   This is the username used to run the server. The `root` and `nobody` users are common choices, but others are possible as well. As a general rule, you should run servers with a low-privilege user

whenever possible as a security precaution. Some servers require root access, though. Consult the server's documentation for details.

Server name   This is the filename of the server. In the preceding example, the server is specified as /usr/sbin/tcpd, which is the TCP Wrappers binary. This program provides some security checks, enabling you to restrict access to a server based on the origin and other factors. Chapter 7, "Documentation and Security," covers TCP Wrappers in more detail.

Parameters   Everything after the server name consists of parameters that are passed to the server. If you use TCP Wrappers, you pass the name of the true target server (such as /usr/sbin/in.ftpd) in this field, along with its parameters.

The hash mark (#) is a comment symbol for /etc/inetd.conf. Therefore, if a server is running via inetd and you want to disable it, you can place a hash mark at the start of the line. If you want to add a server to inetd.conf, you'll need to create an entry for it. Most servers that can be run from inetd include sample entries in their documentation. Many distributions ship with inetd.conf files that include entries for common servers as well, although many of them are commented out; remove the hash mark at the start of the line to activate the server.

After modifying inetd.conf, you must restart the inetd super server itself. This super server normally runs as a standard SysV server, so you can restart it by typing something similar to the following:

```
# /etc/rc.d/init.d/inetd restart
```

Alternatively, you can tell inetd to reload its configuration by passing the SysV startup script the reload parameter rather than restart. The restart option shuts down the server and then starts it again. When you use reload, the server never stops running; it just rereads the configuration file and implements any changes. As a practical matter, the two are quite similar. Using restart is more likely to correctly implement changes, but it's also more likely to disrupt existing connections.

Instead of using the SysV startup scripts, you can use kill or killall (described in Chapter 1) to pass the SIGHUP signal to inetd. This signal causes many servers, including inetd, to reload their configuration files. For instance, you might type kill -HUP *pid* if you know the process ID (PID) of inetd, or killall -HUP inetd to have all instances of inetd reload their configuration files. (Ordinarily, only one instance of inetd runs on a system.) In practice, this should work very much like the reload option to the SysV startup script—in fact, such scripts often use this technique to implement this option.

> It's generally wise to disable as many servers as possible in inetd.conf (or the xinetd configuration files, if you use xinetd). As a general rule, if you don't understand what a server does, disable it. This will improve the security of your system by eliminating potentially buggy or misconfigured servers from the equation.

EXERCISE 9.2

Adding a Server to *inetd*

This exercise investigates configuring inetd to handle a new server. (Distributions that use xinetd typically ship xinetd configuration files for a server in the server's package.) To add a server to inetd, follow these steps:

1. Log into the Linux system as a normal user.

2. Launch an xterm from the desktop environment's menu system, if you used a GUI login method.

3. Acquire root privileges. You can do this by typing **su** in an xterm, by selecting Session ➢ New Root Console from a Konsole, or by using sudo (if it's configured) to run the commands in the following steps.

4. Load the /etc/inetd.conf file into your favorite editor.

5. Look for an existing line in the inetd.conf file for your server. Most default inetd.conf files include sample lines for many popular servers. If such a line is present, comment it out by deleting the hash mark (#) at the start of the line and skip ahead to step 7.

6. Add a line like the following to the file:

```
telnet stream tcp nowait telnetd.telnetd /usr/sbin/tcpd /usr/sbin/in.telnetd
```

This line runs the /usr/sbin/in.telnetd server via TCP Wrappers (/usr/sbin/tcpd). The first column is the server's name, as listed in /etc/services; it determines the port on which inetd is to listen on behalf of the server. The second, third, and fourth entries in the line contain server-specific information. The values shown here are typical, but you should consult a server's documentation to learn what to enter. The fifth entry (telnetd.telnetd) is the account and group that's to own the running process.

7. Save the file and exit from the editor.

8. Type **/etc/rc.d/init.d/inetd reload** to have inetd reread its configuration file and implement the changes. (Some distributions place the script in another location, such as /etc/init.d. Some may not support the reload option, so you may need to use restart instead.)

9. Test your server. In the case of a Telnet server, type **telnet localhost** on the server computer itself, or type **telnet *servername***, where *servername* is the server's hostname or IP address, on another computer.

This exercise uses the Telnet server as an example simply because it's a common server that's likely to be installed on your system. As noted earlier, though, Telnet suffers from serious security problems and so should not be left running on most production systems.

Configuring *xinetd*

The xinetd program is an extended super server. It provides the functionality of inetd plus security options that are similar to those of TCP Wrappers. Modern versions of Fedora, Mandriva, Red Hat, SuSE, and a few other distributions use xinetd by default. Other distributions may use it in the future. If you like, you can replace inetd with xinetd on any distribution.

The /etc/xinetd.conf file controls xinetd. On distributions that use xinetd by default, though, this file contains only global default options and a directive to include files stored in /etc/xinetd.d. Each server that should run via xinetd then installs a file in /etc/xinetd.d with its own configuration options.

Whether the entry for a service goes in /etc/xinetd.conf or a file in /etc/xinetd.d, it contains information similar to that in the inetd.conf file. The xinetd configuration file, though, spreads the information across multiple lines and labels it more explicitly. Listing 9.3 shows an example that's equivalent to the earlier inetd.conf entry. This entry provides precisely the same information as the inetd.conf entry except that it doesn't include a reference to /usr/sbin/tcpd, the TCP Wrappers binary. Because xinetd includes similar functionality, it's generally not used with TCP Wrappers.

> **NOTE**   Chapter 7 covers xinetd security features.

Listing 9.3: Sample xinetd Configuration Entry

```
service ftp
{
        socket_type     = stream
        protocol        = tcp
        wait            = no
        user            = root
        server          = /usr/sbin/in.ftpd
        server_args     = -l
}
```

One additional xinetd.conf parameter is important: disable. If you include the line disable = yes in a service definition, xinetd ignores the entry. Some servers install startup files in /etc/xinetd.d that have this option set by default; you must edit the file and change the entry to read disable = no to enable the server. You can also disable a set of servers by listing their names in the defaults section of the main xinetd.conf file on a line called disabled, as in disabled = ftp shell.

As with inetd, after you make changes to xinetd's configuration, you must restart the super server. You do this by typing a command similar to the one used to restart inetd. As with that command, you can use either reload or restart, with similar effects:

```
# /etc/rc.d/init.d/xinetd restart
```

Also as with `inetd`, you may pass the `SIGHUP` signal to `xinetd` via the `kill` or `killall` command to have it reload its configuration file. This approach may be preferable if you're using a distribution, such as Slackware, that doesn't use a conventional SysV startup script to launch `xinetd`.

Configuring Printing

Printing in Linux is a cooperative effort involving several tools. A system administrator must be familiar with what each of the tools in this collection does, as well as how they interact. As with many other programs that are part of Linux, some of these tools have several versions, which can lead to confusion or incompatibilities if you're not aware of how the system as a whole functions. The basic Linux printing architecture is the same in all cases. One key component of this architecture is the presence of PostScript printers or the use of a program called Ghostscript to convert PostScript into a format that the printer can understand. Whether you use PostScript or non-PostScript printers, one of two broad classes of printing systems is common in Linux: the traditional BSD Line Printer Daemon (LPD) system or one of its derivatives, or the newer Common Unix Printing System (CUPS) utility. In either case, the commands used to print files and to monitor print jobs are similar across all systems, which minimizes problems for end users moving between systems using different printing systems.

The Linux Printing Architecture

Linux printing is built around the concept of a *print queue*. This is a sort of holding area where files wait to be printed. A single computer can support many distinct print queues. These frequently correspond to different physical printers, but it's also possible to configure several queues to print in different ways to the same printer. For instance, you might use one queue to print single-sided and another queue for double-sided printing on a printer that supports duplexing.

Users submit print jobs by using a program called `lpr`. Users can call this program directly, or they may let another program call it. In either case, `lpr` sends the print job into a specified queue. This queue corresponds to a directory on the hard disk, typically in a subdirectory of the `/var/spool/lpd` or `/var/spool/cups` directory. The traditional Linux printing tool is called `lpd`; it runs in the background watching for print jobs to be submitted. A shift to use of another printing system, CUPS, is nearly complete, but it works in a similar manner, at least when considered broadly. Whatever it's called, the printing system accepts print jobs from `lpr` or from remote computers, monitors print queues, and serves as a sort of "traffic cop," directing print jobs in an orderly fashion from print queues to printers.

> **NOTE** The LPI exam covers the older BSD LPD printing system, but in the real world you're more likely to encounter CUPS, at least on reasonably new systems.

One important and unusual characteristic of Linux printing is that it's highly network oriented. As just noted, Linux printing tools can accept print jobs that originate from remote systems as well as from local ones. In fact, even local print jobs are submitted via network protocols, although they don't normally use network hardware, so even a computer with no network connections can print. In addition to being a server for print jobs, `lpd` or CUPS can function as a client, passing print jobs on to other computers that run the same protocols.

One of the deficiencies of the traditional `lpd` printing system is that it's essentially unidirectional—print jobs originate in an application, which blindly produces PostScript (as described shortly) without knowing anything about the printer to which it's printing. The print queue takes this output and sends it on to the printer, which must deal with it as best it can. There's no way for a Linux application to directly query a printer concerning its capabilities, such as whether it supports multiple paper trays or wide forms. This is one of the deficiencies that CUPS aims to correct. Applications can query CUPS about a printer's capabilities—its paper sizes, whether it supports color, and so on. Support for these features is still rare, but this support is likely to become more common as CUPS takes over as the standard Linux printing software.

One confusing aspect of Linux printing is that Linux supports several competing printing systems. In the past, the two most popular have been the original Berkeley Standard Distribution (BSD) LPD printing system and the newer LPRng package. Both work according to the outline just presented, but they differ in some details, some of which are described in the upcoming sections. Between 2001 and 2003, most distributions switched to CUPS as their primary printing systems, although many distributions give a choice of which printing system to use. Even for those distributions that don't give you a choice at installation time, you can rip out one printing system and install another if you like.

Understanding PostScript and Ghostscript

If you've configured printers under Windows, Mac OS, OS/2, or certain other OSs, you're probably familiar with the concept of a *printer driver*. In these OSs, the printer driver stands between the application and the printer queue. In Linux, the printer driver is part of Ghostscript (`http://www.cs.wisc.edu/~ghost/`), which exists as part of the printer queue, albeit a late part. This relationship can be confusing at times, particularly because not all applications or printers need Ghostscript. Ghostscript serves as a way to translate PostScript, a common printer language, into forms that can be understood by many different printers. Understanding Ghostscript's capabilities, and how it fits into a printer queue, can be important for configuring printers.

PostScript: The De Facto Linux Printer Language

Laser printers began to become popular in the 1980s. The first laser printers were very expensive devices, and many of them supported what was at that time a new and powerful printer language: *PostScript*. PostScript printers became quite popular as accessories for the Unix systems of the day. Unix print queues were not designed with Windows-style printer drivers in mind, so Unix programs that took advantage of laser printer features were typically written to produce PostScript output directly. As a result, PostScript developed into the de facto printing standard

for Unix and, by inheritance, Linux. Where programs on Windows systems were built to interface with the Windows printer driver, similar programs on Linux generate PostScript and send the result to the Linux printer queue.

A few programs violate this standard. Most commonly, many programs can produce raw text output. Such output seldom poses a major problem for modern printers, although some PostScript-only models choke on raw text. Some other programs can produce either PostScript or *Printer Control Language (PCL)* output for Hewlett-Packard laser printers or their many imitators. A very few programs can generate output that's directly accepted by other types of printers.

The problem with PostScript as a standard is that it's uncommon on the low- and mid-priced printers with which Linux is often paired. Therefore, to print to such printers using traditional Unix programs that generate PostScript output, you need a translator and a way to fit that translator into the print queue. This is where Ghostscript fits into the picture.

Ghostscript: A PostScript Translator

When it uses a traditional PostScript printer, a computer sends a PostScript file directly to the printer. PostScript is a programming language, albeit one that's oriented toward the goal of producing a printed page as output. As a result, a PostScript printer needs a fair amount of RAM and CPU power. In fact, in the 1980s it wasn't uncommon for PostScript printers to have more RAM and faster CPUs than the computers to which they were connected. Today, though, printers frequently have little RAM and anemic CPUs—particularly on inexpensive ink-jet models.

Ghostscript is a PostScript interpreter that runs on a computer, offloading some of the need for RAM and CPU power. It takes PostScript input, parses it, and produces output in any of dozens of different bitmap formats, including formats that can be accepted by many non-PostScript printers. This makes Ghostscript a way to turn many inexpensive printers into Linux-compatible PostScript printers at very low cost. Ghostscript is available as open source software (GNU Ghostscript), with a more advanced variant (Aladdin Free Public License, or AFPL, Ghostscript) available for free. AFPL Ghostscript is not freely redistributable in any commercial package, though. Because all Linux distributions are available on CD-ROMs sold for a price, they ship with the older GNU Ghostscript, which works well enough for most users.

One of Ghostscript's drawbacks is that it produces large output files. A PostScript file that produces a page filled with text may be just a few kilobytes in size. If this page is to be printed on a 600 dots per inch (dpi) printer using Ghostscript, the resulting output file could be as large as 4MB—assuming it's black and white. If the page includes color, the size could be much larger. In some sense, this is unimportant because these big files will be stored on your hard disk for only brief periods of time. They do still have to get from the computer to the printer, though, and this process can be slow. Also, some printers (particularly laser printers) may require memory expansion to operate reliably under Linux.

Squeezing Ghostscript into the Queue

Printing to a non-PostScript printer in Linux requires fitting Ghostscript into the print queue. This is generally done through the use of a *smart filter*. This is a program that's called as part of the printing process. The smart filter examines the file that's being printed, determines its type, and passes the file through one or more additional programs before the printing software

---

🌐 Real World Scenario

Choosing an Appropriate Printer for Linux

If you want a speedy printer for Linux, choose a model with built-in PostScript. This is particularly true for textual and line-art output, which suffers the most in terms of size expansion going from Post-Script to bitmap. In my experience, Ghostscript-driven printers work well enough for 600dpi black-and-white printers with speeds of up to about six pages per minute (ppm), although theoretically both the parallel port and USB 1.x port should be able to handle speeds of three to five times that value. If the printer's speed is greater than that, the parallel or USB 1.x port may not be able to deliver the necessary performance, although you may be able to tweak it to get somewhat better speed.

Color ink-jet printers are generally limited more by the speed of the print head than by the speed of the data coming over their ports. Few such printers directly support PostScript, either. Some models come with Windows-based PostScript engines that are conceptually similar to Ghostscript, but such software is useless under Linux. There are a few PostScript ink-jets on the market, as well as color PostScript printers that use other printing technologies.

For information on what printers are supported by Ghostscript, check the Ghostscript web page or the GNU/Linux Printing web page (`http://www.linuxprinting.org/printer_list.cgi`).

---

sends it on to the printer. The smart filter can be configured to call Ghostscript with whatever parameters are appropriate to produce output for the queue's printer.

When your system uses BSD LPD or LPRng, the smart filter is specified with the `if` field in `/etc/printcap`, as described shortly, in "Configuring the `/etc/printcap` File." Several smart filter packages are available for Linux, including `rhs-printfilters` (used in older Red Hat distributions and some of its derivatives), Apsfilter (used in several other distributions), and magicfilter. CUPS ships with its own set of smart filters, which it calls automatically when you tell the system what model printer you're using.

Configuration of the smart filter can be tricky, but most distributions include setup tools that help immensely. The upcoming section "Using a Configuration Tool" describes the use of one such tool for BSD LPD or LPRng printing systems. CUPS uses its own Web-based configuration tool, as described in the upcoming section "Using the Web-Based CUPS Utilities." I highly recommend that you use such programs when configuring your system to print.

The end result of a typical Linux printer queue configuration is the ability to treat any supported printer as if it were a PostScript printer. Applications that produce PostScript output can print directly to the queue. The smart filter detects that the output is PostScript and runs it through Ghostscript. The smart filter can also detect other file types, such as plain-text and various graphics files, and it can send them through appropriate programs instead of or in addition to Ghostscript in order to create a reasonable printout.

If you have a printer that can process PostScript itself, the smart filter is usually still involved, but it doesn't pass PostScript through Ghostscript. In this case, the smart filter passes PostScript directly to the printer, but it still sends other file types through whatever processing is necessary to turn them into PostScript.

Running a Printing System

Because Linux printing systems run as daemons, they must be started before they're useful. This task is normally handled automatically via startup scripts in /etc/rc.d or /etc/rc?.d (where *?* is a runlevel number). Look for startup scripts that contain the strings lpd, lprng, or cups in their names to learn what your system is running. If you're unsure if a printing system is currently active, use the ps utility to search for running processes by these names, as in:

```
$ ps ax | grep cups
 3713 ?        S        0:00 cupsd
```

This example shows that cupsd, the CUPS daemon, is running, so the system is using CUPS for printing. If you can't find any running printing system, consult your distribution's documentation to learn what is available and check that the appropriate package is installed. All major distributions include startup scripts that should start the appropriate printing daemon when the computer boots.

Configuring BSD LPD and LPRng

Fortunately, basic printer configuration for both the original BSD printing tools and LPRng is similar. You can configure everything by hand by directly editing configuration files, but certain critical details—namely, how your smart filter is set up—differ from one distribution to another, and they can be tedious to locate. Therefore, direct file editing is best reserved for cases where you can forgo the smart filter or if you're willing to track down the documentation for whatever smart filter your system uses. In most cases, it's easier to use a configuration tool to do the initial printer configuration, and then you can tweak that configuration by hand if necessary. Either way, the printing daemon runs in the background and accepts print jobs submitted via lpr.

Configuring the */etc/printcap* File

The /etc/printcap file is at the heart of both the BSD and LPRng printing systems. Listing 9.4 illustrates the format of /etc/printcap by showing an entry for a single printer. You can define multiple printers in /etc/printcap; just be sure to use different names.

Listing 9.4: A Sample */etc/printcap* File

```
lp|hp4000:\
        :lp=/dev/lp0:\
        :br#57600:\
        :rm=:\
        :rp=:\
        :sd=/var/spool/lpd/lp:\
        :mx#0:\
        :sh:\
        :if=/var/spool/lpd/lp/printfilter:
```

Technically, each printer definition is one line long. The `/etc/printcap` entries, however, traditionally make heavy use of the common Linux convention of using a backslash (\) to signal a line continuation. (Note that every line in Listing 9.4 *except* the last one ends in a backslash.) This makes the printer definitions easier to read. Each component within the `/etc/printcap` entry is separated from the others by colons (`:`). Common components of a print queue definition include the following:

Printer name   Each printer definition begins with one or more names for the printer. If the printer has multiple names, they're separated from each other by vertical bars (`|`). Traditionally, the default printer is called `lp`. Listing 9.4's example expands on this by adding the name `hp4000`. Users may print using either name with the same results.

Printer device filename   The `lp=/dev/lp0` entry defines the device filename for the printer. In the case of Listing 9.4, the printer device is `/dev/lp0`, which corresponds to the first parallel port. Many modern printers support USB interfaces, which use the `/dev/usb/lp`*n* devices, where *n* is a number from 0 up. A few printers use the old RS-232 serial ports, which may be accessed as `/dev/ttyS`*n*. This entry may be omitted if the printer is shared from another computer.

Baud rate   The `br` parameter's name stands for "baud rate"; it defines the communications speed for RS-232 serial printers. This option is normally omitted for parallel port, USB, and network printers. It doesn't do any harm to leave it in, however, as in Listing 9.4.

Remote machine   If you're defining a printer queue for a printer that's connected to another computer or that's connected directly to the network, you specify its machine name with the `rm` option. Like `br`, it can be omitted if not used.

Remote print queue   The `rp` option is used in conjunction with `rm`, but it specifies the name of the print queue on the remote system. For instance, your local `epson` queue might print to a queue called `inkjet` on a remote system. Your local users will use the name `epson`, and your `lpd` will pass the job on to the remote system's `lpd`, which will print to the remote `inkjet` queue. This option may be omitted if the printer is local, but leaving it blank in this case (as in Listing 9.4) does no harm.

> Although you *can* use different local and remote names, using the same name for both will help avoid confusion.

Spool directory   The `sd` parameter name stands for *spool directory*. This is the location of the print queue on your hard disk. By convention, this is a subdirectory of the `/var/spool/lpd` directory, named after the print queue's primary name. If you create print queues by hand, you'll need to create this directory. It should normally have fairly restrictive permissions (such as `rwx------` and ownership by `root`) so that people can't read or delete each other's print jobs.

Maximum print job size   The `mx` option sets the maximum size of a print job, in bytes. You can use this option to restrict abuses, but be aware that the print job size in bytes and its size in pages are poorly correlated. If it is set to `0`, there's no limit on job size. This option uses a hash mark (`#`) rather than an equal sign (`=`) to set its value.

Suppress header   The sh option takes no value. It stands for suppress header, and if it's present, Linux does *not* print a header page with information on the user who printed the job. This configuration makes sense for workstations, but on multiuser systems and print servers, omitting the sh option and using the resultant headers can help you organize printouts from multiple users.

Input filter   This option sets the input filter filename, which is part of the smart filter associated with the queue. This is frequently a script located within the spool directory. The script sets various options (such as the name of the Ghostscript driver to be used) and calls the smart filter files located elsewhere on the disk.

The /etc/printcap file is fairly complex and supports many options. You can learn about more of them from the file's man page (type **man printcap**). The preceding options cover what you're likely to do with the queue, however, aside from smart filter configuration.

After reconfiguring your print queues, you may need to restart your printer daemon. On most systems, you can do this by passing the restart parameter to the LPRng or BSD LPD printing startup script. The following is an example of how this might be done:

```
# /etc/rc.d/init.d/lpd restart
```

The exact name and location of this file will vary from one distribution to another. You should use this command only when your system isn't actively printing.

Using a Configuration Tool

Much of the challenge of printing in Linux doesn't come from the /etc/printcap file; it comes from telling the system about your printer—that is, smart filter and Ghostscript configuration. GNU Ghostscript comes standard with all major Linux distributions, and it is probably adequate for your system. In a few cases, though, you may need to upgrade to the more recent AFPL Ghostscript or obtain a version with some unusual drivers compiled into it. The GNU/Linux Printing web page (http://www.linuxprinting.org/printer_list.cgi) can be an extremely useful resource in tracking down appropriate drivers.

In most cases, the easiest way to configure a print queue with a smart filter for a specific non-PostScript printer is to use a printer configuration tool. Most major Linux distributions come with these tools, although which ones come with which distribution varies substantially. This section describes the use of one popular printer configuration tool, Apsfilter (http://www.apsfilter.org).

> **NOTE**   Because most distributions now use CUPS as their primary printing system, distribution-specific tools frequently handle CUPS configuration, not BSD LPD or LPRng configuration. Apsfilter is distribution neutral but ships with many distributions to handle both BSD LPD and LPRng configuration. The package also includes a smart filter that it inserts into print queues to detect PostScript and other file types.

To use Apsfilter for printer configuration, you must first install the package. Look for it on your distribution's installation media or download and install it from the Apsfilter web page. If Apsfilter doesn't ship with your distribution and you download it from the website, you'll need to compile and install the software; consult the Apsfilter documentation for details. Apsfilter configuration involves following a series of configuration steps. Begin by typing **/usr/share/apsfilter/SETUP** as `root`. This action launches the text-based configuration tool. (Some distributions put the SETUP program in another directory, so consult your distribution's documentation if you can't find this program.)

The Apsfilter SETUP program requires you to identify your printer model, set the port or remote queue the printer is to use, set the paper size, and so on. Before you exit from this program, be sure to use its test option; this should produce a legible test printout to verify that the configuration works. If it doesn't produce output or if the system prints reams of gibberish, you should review your configuration.

> Gibberish as output usually indicates that you've selected the wrong printer. No output at all could mean an incorrect printer selection, an incorrect printer device file selection, or various other problems.

If your system uses another printer configuration tool, such as magicfilter, the configuration details will differ, but you must still enter the same basic information. Consult the filter's documentation for details.

> All magic filter programs store configuration files somewhere. For Apsfilter, the location is /etc/apsfilter, with the apsfilterrc file holding the actual configurations. For magicfilter, it's /etc/magicfilter. You can edit these files directly, but doing so can be tedious.

You can create several print queues, even if you have just one printer. For instance, you might create one queue that prints at high resolution and another that prints at a lower resolution. Your users can then pick the desired print resolution by choosing a different print queue, as described shortly—for instance, you might call one `epson360` and the other `epson720`. You can even create one queue that prints with the normal print filters and another that doesn't use a filter—that is, a raw queue. A raw print queue can be useful if you have programs that can print directly to the printer type you use. For instance, the GIMP includes drivers for many specific printer models, and so they can do without Ghostscript in many cases.

Configuring CUPS

Although CUPS plays the same role in Linux as BSD LPD or LPRng, and uses Ghostscript in a conceptually similar way, CUPS configuration is quite different from BSD LPD or LPRng configuration. CUPS doesn't rely on an `/etc/printcap` file (although it generates a simple one on-the-fly for the benefit of programs that refer to it to learn what printers are available). Instead, it uses various configuration files in the `/etc/cups` directory and its subdirectories. You can

edit these files directly, and you may need to do so if you want to share printers or use printers shared by other CUPS systems. The simplest way to add printers to CUPS, though, is to use the tool's Web-based configuration utility.

> **NOTE** CUPS isn't mentioned in the LPI objectives, but it's now the dominant printing system for Linux. Thus, you should know how to deal with CUPS.

Editing the CUPS Configuration Files

You can add or delete printers by editing the /etc/cups/printers.conf file, which consists of printer definitions. Each definition begins with the name of a printer, identified by the string DefaultPrinter (for the default printer) or Printer (for a nondefault printer) in angle brackets (<>), as in the following:

```
<DefaultPrinter okidata>
```

This line marks the beginning of a definition for a printer queue called okidata. The end of this definition is a line that reads </Printer>. Intervening lines set assorted printer options, such as identifying strings, the printer's location (its local hardware port or network location), its current status, and so on. Additional options are stored in a *PostScript Printer Definition (PPD)* file that's named after the queue and stored in the /etc/cups/ppd subdirectory. PPD files follow an industry-standard format. For PostScript printers, you can obtain a PPD file from the printer manufacturer, typically from a driver CD-ROM or from the manufacturer's website. CUPS and its add-on driver packs also ship with a large number of PPD files that are installed automatically when you use the Web-based configuration utilities.

As a general rule, you're better off using the CUPS Web-based configuration tools to add printers rather than adding printers by directly editing the configuration files. If you like, though, you can study the underlying files and tweak the configurations using a text editor to avoid having to go through the full Web-based tool to make a minor change.

One exception to this rule relates to configuring the CUPS Web-based interface tool itself and CUPS's ability to interface with other CUPS systems. One of the great advantages of CUPS is that it uses a new network printing protocol, known as the *Internet Printing Protocol (IPP),* in addition to the older LPD protocol used by BSD LPD and LPRng. IPP supports a feature it calls browsing, which enables computers on a network to automatically exchange printer lists. This feature can greatly simplify configuring network printing. You may need to change some settings in the main CUPS configuration file, /etc/cups/cupsd.conf, to enable this support.

The /etc/cups/cupsd.conf file, which is structurally similar to the Apache web server configuration file, contains a number of configuration blocks that specify which other systems should be able to access it. Each block controls access to a particular location on the server. These blocks look like this:

```
<Location /printers>
Order Deny,Allow
Deny from All
BrowseAllow from 127.0.0.1
```

```
BrowseAllow from 192.168.1.0/24
BrowseAllow from @LOCAL
Allow from 127.0.0.1
Allow from 192.168.1.0/24
Allow from @LOCAL
</Location>
```

> **NOTE** If you're configuring a workstation with a local printer that you don't want to share, or if you want to configure a workstation to use printers shared via LPD or some other non-IPP printing protocol, you shouldn't need to adjust /etc/cups/cupsd.conf. If you want to access remote IPP printers, however, you should at least activate browsing by setting the directive Browsing On, as described shortly. You shouldn't have to modify your location definitions unless you want to share your local printers.

The `/printers` location, shown here, controls access to the printers themselves. The following list includes features of this example:

Directive order   The `Order Deny,Allow` line tells CUPS in which order it should apply allow and deny directives—in this case, allow directives modify deny directives.

Default policy   The `Deny from All` line tells the system to refuse all connections except those that are explicitly permitted.

Browsing control lines   The `BrowseAllow` lines tell CUPS from which other systems it should accept browsing requests. In this case, it accepts connections from itself (127.0.0.1), from systems on the 192.168.1.0/24 network, and from systems connected to local subnets (`@LOCAL`).

Access control lines   The `Allow` lines give the specified systems non-browse access to printers— that is, those systems can print to local printers. In most cases, the `Allow` lines will be the same as the `BrowseAllow` lines.

You can also create a definition that uses `Allow from All` and then creates `BrowseDeny` and `Deny` lines to limit access. As a general rule, though, the approach shown in this example is safer. Locations other than the `/printers` location can also be important. For instance, there's a root (/) location that specifies default access permissions to all other locations and an `/admin` location that controls access to CUPS administrative functions.

Before the location definitions in `cupsd.conf` are a few parameters that enable or disable browsing and other network operations. You should look for the following options specifically:

Enabling browsing   The `Browsing` directive accepts `On` and `Off` values. The CUPS default is to enable browsing (`Browsing On`), but some Linux distributions disable it by default.

Browsing access control   The `BrowseAddress` directive specifies the broadcast address to which browsing information should be sent. For instance, to broadcast data on your printers to the 192.168.1.0/24 subnet, you'd specify `BrowseAddress 192.168.1.255`.

Once you've configured a CUPS server to give other systems access to its printers via appropriate location directions, and once you've configured the client systems to use browsing via

`Browsing On`, all the systems on the network should auto-detect all the printers on the network. There's no need to configure the printer on any computer except the one to which it's directly connected. All printer characteristics, including their network locations and PPD files, are propagated automatically by CUPS. This feature is most important in configuring large networks with many printers or networks on which printers are frequently added and deleted.

Obtaining CUPS Printer Definitions

The basic version of CUPS ships with smart filter support for just a few printers, including raw queues that do no processing and a few models from Hewlett-Packard, Epson, and Okidata. If you use another printer, you should obtain extra CUPS printer definitions. These definitions may consist of PPD files, appropriate behind-the-scenes "glue" to tell CUPS how to use them, and possibly Ghostscript driver files. These printer definitions can be obtained from several sources:

Your Linux distribution   Many distributions ship extra printer definitions in a package called `cups-drivers` or something similar, so check your distribution for such a package. In truth, this package is likely to be the Foomatic or GIMP Print package under another name.

Foomatic   The Linux Printing website hosts a set of utilities and printer definitions known collectively as Foomatic (`http://www.linuxprinting.org/foomatic.html`). These provide many additional printer definitions for CUPS (as well as for other printing systems).

GIMP Print   The GNU Image Manipulation Program (GIMP) is a major Linux bitmap graphics program that supports its own printer drivers. These in turn have been spawned off into a package called GIMP Print, which can be integrated with CUPS to provide additional printer options. Check `http://gimp-print.sourceforge.net` for more information.

ESP Print Pro   Easy Software Products (ESP) is the company that first developed CUPS. Although CUPS is open source, ESP offers a variety of printer definitions for CUPS for a price. See `http://www.easysw.com/printpro/` for more details.

   If you're printing to one of the basic printers supported by the standard CUPS definitions, you may not need to add anything else. You might also find that your distribution has installed a set of definitions as part of the main CUPS package or in an add-on package, such as `cups-drivers`, without explicit instruction. In either of these cases, you're set and need not do anything else. If you start configuring printers and can't find your model, though, you should look for an additional printer definition set from one of the preceding sources.

Using the Web-Based CUPS Utilities

The CUPS IPP printing system is closely related to the Hypertext Transfer Protocol (HTTP) used on the Web. The protocol is so similar, in fact, that you can access a CUPS daemon by using a web browser. You need only specify that you want to access the server on port 631—the normal printer port. To do so, enter **`http://localhost:631`** in a web browser on the computer running CUPS. (You may be able to substitute the hostname or access CUPS from another computer by using the other computer's hostname, depending on your `cupsd.conf` settings.) This action brings up a list of administrative tasks you can perform. Click Manage Printers to open the printer management page, as shown in Figure 9.4.

FIGURE 9.4   CUPS provides its own Web-based configuration tool.
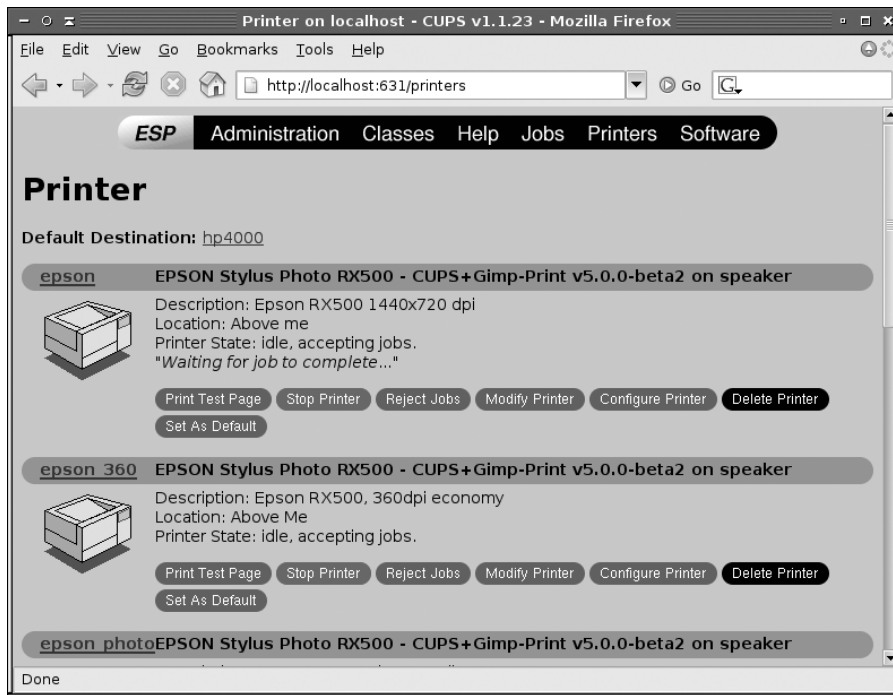
> **NOTE** If you're configuring a stand-alone computer or the only one on a network to use CUPS, the printer list will be empty, unlike the one shown in Figure 9.4. If other computers on your network use CUPS, you may see their printers in the printer list, depending on their security settings.

You can add, delete, or modify printer queues using the CUPS Web control system. To add a printer, follow these steps:

1. Scroll to the bottom of the page and click Add Printer. (This option isn't visible in Figure 9.4 because it's too far down the existing printer list.) You're likely to be asked for a username and password.

2. Type **root** as the username and the administrative password as the password, and then click OK.

> **WARNING** CUPS doesn't normally encrypt its data, so you shouldn't use it to administer printers remotely. Doing so would be a security risk, as the passwords would be exposed to sniffing.

3. The system displays a page asking for the printer's name, location, and description. Enter appropriate information in the Name, Location, and Description fields. These fields are all entirely descriptive, so enter anything you like. (Users will use your entry in the Name field to access the printer, though.) When you click Continue, CUPS asks for the printer device.

4. The printer device may be a local hardware port (such as a parallel printer port or a USB port), a remote LPD printer, a remote SMB/CIFS (Samba) printer, or other devices. The precise options available vary from one distribution to another. Select the appropriate one from the pop-up list and click Continue.

> **NOTE** USB printers must be turned on and connected to the computer when CUPS starts in order to be configured or used. If your printer was turned off or disconnected when your system booted, you may need to restart CUPS via its SysV startup script and then start over again to configure it.

5. If you entered a network printer, the result is a page in which you enter the complete path to the device. Type the path, such as **lpd://printserv/brother** to print to the brother queue on the printserv computer. Click Continue when you're done.

6. If you entered a local device in step 4 or after you've entered the complete path in step 5, you'll see a list of driver classes, such as PostScript and HP. Select one and click Continue.

7. CUPS now displays a complete list of printer models within the class you selected in step 6. Select an appropriate model and click Continue.

8. CUPS informs you that the printer has been added.

If you click the Printers item at the top of the page, you should be returned to the printers list (Figure 9.4), but your new printer should be listed among the existing queues. You can print a test page by clicking Print Test Page. If all goes well, a test page will emerge from your printer. If it doesn't, go back and review your configuration by clicking Modify Printer. This action takes you through the steps for adding a printer but with your previous selections already entered as the defaults. Try changing some settings until you get the printer to work.

From the printer queue list, you can also click Configure Printer to set various printer options. What options are available depends on the printer, but common options include the resolution, color dithering options, the paper size, whether or not to enable double-sided printing, and the presence of banner pages.

Printing to Windows or Samba Printers

If your network hosts many Windows computers, you may use the Server Message Block/Common Internet File System (SMB/CIFS) for file and printer sharing among Windows systems. Linux's Samba server also implements this protocol and so can be used for sharing printers from Linux.

> **NOTE** Chapter 10 describes the basics of configuring a Linux Samba server, so consult it if you want to share an existing printer queue with Windows clients.

On the flip side, you can print to an SMB/CIFS printer queue from a Linux system. To do so, you select an SMB/CIFS queue in the printer configuration tool (such as Apsfilter or the CUPS configuration option). Apsfilter lists the SMB/CIFS queue option as `Windows / NT (samba)`. Under CUPS, it's called Windows Printer via SAMBA in step 4 in the preceding procedure.

Precisely how you proceed with configuration depends on the tool you're using. Most guide you through the process by asking for a hostname, share name, username, and password. Some servers enable you to omit the username or password or to enter random values for these fields. Others require all four components to work. For CUPS, you must provide this information, but the format is not obvious from the Web-based configuration tool:

`smb://username:password@SERVER/SHARE`

This is a uniform resource identifier (URI) for an SMB/CIFS share. You must substitute appropriate values for `username`, `password`, `SERVER`, and `SHARE`, of course. Once this is done and you've finished the configuration, you should be able to submit print jobs to the SMB/CIFS share.

> **NOTE** SMB/CIFS printers hosted by Windows systems are usually non-PostScript models, so you must select a local Linux smart filter and Ghostscript driver, just as you would for a local printer. Printers hosted by Linux systems running Samba, though, are frequently configured to act like PostScript printers, so you should select a PostScript driver when connecting to them.

In practice, you may be faced with a decision: Should you use LPD, IPP, or SMB/CIFS for submitting print jobs? To be sure, not all print servers support all three protocols, but a Linux server might support them all. As a general rule, IPP is the simplest to configure because it supports browsing, which means that CUPS clients shouldn't need explicit configuration to handle specific printers. This makes IPP the best choice for Linux-to-Linux printing, assuming both systems run CUPS. When CUPS isn't in use, LPD is generally easier to configure than SMB/CIFS, and it has the advantage of not requiring the use of a username or password to control access. Because SMB/CIFS security is password-oriented, clients typically store passwords in an unencrypted form on the hard disk. This fact can become a security liability, particularly if you use the same account for printing as for other tasks. On the other hand, sometimes use of a password on the server provides more of a security benefit than the risk of storing that password on the client. Generally speaking, if clients are few and well protected, while the server is exposed to the Internet at large, using passwords can be beneficial. If clients are numerous and exposed to the Internet while the print server is well protected, though, a password-free security system that relies on IP addresses may be preferable.

Monitoring and Controlling the Print Queue

Several utilities can be used to submit print jobs and to examine and manipulate a Linux print queue. These utilities are `lpr`, `lpq`, `lprm`, and `lpc`. All of these commands can take the `-P` parameter to specify that they operate on a specific print queue.

Printing Files with *lpr*

Once you've configured the system to print, you probably want to start printing. As mentioned earlier, Linux uses the `lpr` program to submit print jobs. This program accepts many options that you can use to modify the program's action:

Specify a queue name   The `-P`*queuename* option enables you to specify a print queue. This is useful if you have several printers or if you've defined several queues for one printer. If you omit this option, the default printer is used.

> **NOTE**   In the original BSD version of `lpr`, there should be no space between the `-P` and the *queuename*. LPRng and CUPS are more flexible in this respect; you can insert a space or omit it as you see fit.

Delete original file   Normally, `lpr` sends a copy of the file you print into the queue, leaving the original unharmed. Specifying the `-r` option causes `lpr` to delete the original file after printing it.

Suppress banner   The `-h` option suppresses the banner for a single print job. It's not available in CUPS.

Job name specification   Print jobs have names to help identify them, both while they're in the queue and once printed (if the queue is configured to print banner pages). The name is normally the name of the first file in the print job, but you can change it by including the `-J` *jobname* option.

User e-mail notification   The `-m` *username* option causes `lpd` to send e-mail to *username* when the print job is complete. It's not available in CUPS.

Number of copies   You can specify the number of copies of a print job by including the number after a dash, as in `-3` to print three copies of a job.

Suppose you have a file called `report.txt` that you want to print to the printer attached to the `lexmark` queue. This queue is often quite busy, so you want the system to send e-mail to your account, `ljones`, when it's done so that you know when to pick up the printout. You could use the following command to accomplish this task, at least with the BSD LPD and LPRng versions of `lpr`:

```
$ lpr -Plexmark -m ljones report.txt
```

The `lpr` command is accessible to ordinary users as well as to `root`, so anybody may print using this command. It's also called from many programs that need to print directly, such as graphics programs and word processors. These programs typically give you some way to adjust the print command so that you can enter parameters such as the printer name. For instance, Figure 9.5 shows Mozilla Firefox's Print dialog box, which features a pop-up list button that lets you select the print queue. This is the Printer field in Figure 9.5, but some programs call it something else. Some programs also provide a text entry field in which you type some or all of an `lpr` command instead of selecting from a pop-up list of available queues. Consult the program's documentation if you're not sure how it works.

FIGURE 9.5   Most Linux programs that can print do so by using `lpr`, but many hide the details of the `lpr` command behind a dialog box.
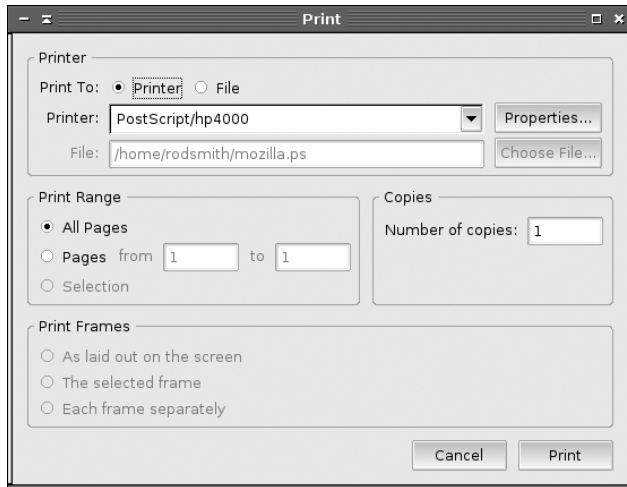
Sometimes you want to process a file in some way prior to sending it to the printer. Chapter 1 covers some commands that can do this, such as `fmt` and `pr`. Another handy program is `mpage`, which reads plain-text or PostScript files and reformats them so that each printed sheet contains several reduced-size pages from the original document. This can be a good way to save paper if you don't mind a reduction in the document size. In the simplest case, you can use `mpage` much as you'd use `lpr`:

```
$ mpage -Plexmark report.ps
```

This command prints the `report.ps` file reduced to fit four pages per sheet. You can change the number of source pages to fit on each printed page with the `-1`, `-2`, `-4`, and `-8` options, which specify one, two, four, or eight input pages per output page, respectively. Additional `mpage` options exist to control features such as the paper size, the font to be used for plain-text input files, and the range of input file pages to be printed. Consult the `man` page for `mpage` for more details.

Displaying Print Queue Information with *lpq*

The `lpq` utility displays information on the print queue—how many files it contains, how large they are, who their owners are, and so on. By entering the user's name as an argument, you can also use this command to check on any print jobs owned by a particular user. To use `lpq` to examine a queue, you might issue a command like the following:

```
$ lpq -Plexmark
Printer: lexmark@speaker
 Queue: 1 printable job
 Server: pid 14817 active
```

```
Unspooler: pid 14822 active
Status: printing 'rodsmith@speaker+787', file 1 'Insight.ps', size 672386,
➥format 'l' at 14:57:10
Rank   Owner/ID            Class Job  Files        Size   Time
active  rodsmith@speaker+787  A   787 Insight.ps  672386  14:56:22
```

> **NOTE** This example shows the output of LPRng's lpq. Systems that use the original BSD LPD and CUPS display less information, but the most important information (such as the job number, job owner, job filename, and job size) are present in all cases.

Of particular interest is the job number—787 in this example. You can use this number to delete a job from the queue or reorder it so that it prints before other jobs. Any user may use the lpq command.

Removing Print Jobs with *lprm*

The lprm command removes one or more jobs from the print queue. There are several ways to issue this command:

- If it's issued with a number, that number is understood to be the job ID (as shown in lpq's output) of the job that's to be deleted.
- If root runs lprm and passes a username to the program, it removes all the jobs belonging to that user.
- If a user runs the BSD lprm and passes a dash (-) to the program, it removes all the jobs belonging to the user. LPRng uses all instead of a dash for this purpose.
- If root runs the BSD lprm and passes a dash (-) to the program, it removes all print jobs belonging to all users. Again, LPRng uses all for this purpose.

This program may be run by root or by an ordinary user, but as just noted, its capabilities vary depending on who runs it. Ordinary users may remove only their own jobs from the queue, but root may remove anybody's print jobs.

Controlling the Print Queue with *lpc*

The lpc utility starts, stops, and reorders jobs within print queues. The lpc utility takes commands, some of which require additional parameters. You can pass the printer name with -P, as with other printer utilities, or you can pass this information *without* the -P parameter. In the latter case, the print queue name appears immediately after the command. The following are the most useful actions you can take with lpc:

Abort printing   The abort command stops printing the current job and any other jobs in the queue but leaves those jobs intact. Subsequently issuing the start command will resume printing.

Disable future printing   The disable command sets the queue to reject further print jobs but does *not* halt printing of jobs currently in the queue.

Disable all printing   The `down` command stops printing to the queue and sets the queue to reject further print jobs.

Enable future printing   You can enable the queue so that it begins accepting print jobs again by using the `enable` command. This is the opposite of `disable`.

Exit from the program   If `lpc` is started in interactive mode (described shortly), the `exit` command will terminate this mode, returning you to a shell prompt.

Start printing   The `start` command begins printing and starts an `lpd` process for the queue.

Stop printing   The `stop` command stops `lpd` so further printing is disabled after the current job completes.

Reorder print jobs   The `topq` *jobid* command moves the job whose ID is *jobid* to the start of the queue (after the currently printing document). Use this command to reprioritize your print queue.

Enable all printing   The `up` command enables the specified print queue; this is the opposite of `down`.

Obtain status information   You can display status information on all of the print queues, or on the one selected via -P, by using the `status` command.

> Although CUPS ships with an `lpc` command, its functionality is *very* limited; its only useful command is `status`. The next section, "Controlling CUPS Print Queues," describes some alternatives in CUPS.

You can run `lpc` in interactive mode, in which you issue one command after another, or you can have it execute a single command and then exit by specifying the command on the same command line you use to launch `lpc`. As an example, suppose you want to adjust the printing of job 945 on the `brother` queue (identified through a previous `lpq` command) so that it's next to print. You could issue the following command to do this:

```
# lpc topq brother 945
```

Although ordinary users may run `lpc`, for the most part, they can't do anything with it. Typical `lpc` operations require superuser privileges.

Controlling CUPS Print Queues

The version of `lpc` that ships with CUPS is quite limited, so most of the `lpc` commands just described don't work with it. You can, though, control a CUPS print queue in a couple of ways:

- You can disable a queue by clicking the Stop Printer link for the printer on the CUPS Web interface (Figure 9.4). When you do so, this link changes to read Start Printer, which reverses the effect. The Jobs link also provides a way to cancel and otherwise manage specific jobs.

- You can use a series of commands, such as `enable`, `disable`, `start`, and `stop`, to control the queue. Most of these commands have the same names as their `lpc` command counterparts, but you type them at a `bash` shell prompt. One unusual addition to the list is `lpmove`, which enables you to move a print job from one queue to another. This can be handy if you must shut down a queue for maintenance and want to redirect the queue's existing jobs to another printer.

Summary

Linux is a network-enabled OS, and it relies on its networking features more than do most OSs. This networking is built around TCP/IP, so you should understand the basics of this protocol stack, including IP addresses, hostnames, and routing. Most Linux distributions provide tools to configure networking during system installation, but if you want to temporarily or permanently change your settings, you can do so. Tools such as `ifconfig` and `route` can temporarily change your network configuration, and editing critical files or running distribution-specific utilities enable you to make your changes permanent. If your system isn't connected to a local network, you can use a modem in conjunction with PPP to initiate a connection to the Internet. Most of the same networking tools, including basic diagnostic programs, work on both local networks and PPP connections.

Once your basic network configuration is up and running, you can begin looking into network clients and servers. One important class of server is a super server, which manages other servers. The `inetd` and `xinetd` super servers handle this task in Linux. Another important server is Linux's printing system, which is likely to be BSD LPD, LPRng, or CUPS. These systems all use daemons that use network protocols to accept print jobs, Ghostscript to convert PostScript to printer-specific formats, and assorted tools to submit and manage print jobs.

Exam Essentials

Describe the information needed to configure a computer on a static IP network.   Four pieces of information are important: the IP address, the netmask (aka the network mask or subnet mask), the network's gateway address, and the address of at least one DNS server. The first two are required, but if you omit either or both of the latter two, you won't be able to connect to the Internet or use most DNS hostnames.

Determine when using /etc/hosts over DNS makes the most sense.   The /etc/hosts file provides a static mapping of hostnames to IP addresses on a single computer. As such, maintaining this file on a handful of computers for a small local network is fairly straightforward, but when the number of computers rises beyond a few or when IP addresses change frequently, running a DNS server to handle local name resolution makes more sense.

Summarize tools you can use to translate between hostnames and IP addresses. The nslookup program can perform these translations in both directions using either command-line or interactive modes, but this program has been deprecated. You're better off using host for simple lookups or dig for more complex tasks.

Describe the function of network ports. Network ports enable packets to be directed to specific programs; each network-enabled program attaches itself to one or more ports, sending data from that port and receiving data directed to the port. Certain ports are assigned to be used by specific servers, enabling client programs to contact servers by directing requests at specific port numbers on the server computers.

Explain when you should use static IP addresses or DHCP. Static IP address configurations involve manually entering the IP address and other information and is used when a network lacks a Dynamic Host Configuration Protocol (DHCP) server or when a computer shouldn't be configured by that server (say, because the computer *is* the DHCP server). DHCP configuration is easier to set up on the client but will only work if the network has a DHCP server system.

Explain what the route command accomplishes. The route command displays or modifies the routing table, which tells Linux how to direct packets based on their destination IP addresses.

Summarize the function of PPP. The Point-to-Point Protocol negotiates a TCP/IP connection, typically acquiring requisite information from the PPP server. It's used to connect computers via telephone lines and is used in modified form for some broadband links.

Describe some basic network diagnostic tools. The ping program tests basic network connectivity, and traceroute performs similar but more complex tests that can help you localize where on a route between two systems a problem exists. Packet sniffers such as tcpdump provide detailed information on the network packets "seen" by a computer, which can be a useful way to verify that certain packet types are actually being sent or received.

Identify the purpose of a super server. Super servers, such as inetd and xinetd, manage incoming network connections for multiple servers. They can add security and convenience features and they can help minimize the memory load imposed by seldom-accessed servers.

Describe the role of lpd and CUPS in Linux printing. The line printer daemon (lpd) and CUPS play similar roles. Both accept local and remote print jobs, maintain the local print queue, call smart filters, and pass data to the printer port in an orderly fashion. CUPS is rapidly taking over from lpd as the standard Linux printing system.

Explain the role of Ghostscript in Linux printing. PostScript is the standard Linux printing language, and Ghostscript converts PostScript into bitmap formats that are acceptable to non-PostScript printers. Thus, Ghostscript is a critical translation step in most Linux print queues.

Summarize how print jobs are submitted and managed under Linux. You use lpr to submit a print job for printing, or an application program may call lpr itself or implement its functionality directly. The lpq utility summarizes jobs in a queue, lprm can remove print jobs from a queue, and lpc can otherwise control a print queue.

Review Questions

1. Which types of network hardware does Linux support? (Select all that apply.)

    A. Token Ring

    B. Ethernet

    C. DHCP

    D. Fibre Channel

2. Which of the following is a valid IP address for a computer on a TCP/IP network?

    A. 202.9.257.33

    B. 63.63.63.63

    C. 107.29.5.3.2

    D. 98.7.104.0/24

3. You want to set up a computer on a local network via a static TCP/IP configuration, but you lack a gateway address. Which of the following is true?

    A. Because the gateway address is necessary, no TCP/IP networking functions will work.

    B. TCP/IP networking will function, but you'll be unable to convert hostnames to IP addresses or vice versa.

    C. You'll be able to communicate with machines on your local network segment but not with other systems.

    D. The computer won't be able to tell which other computers are local and which are remote.

4. Which of the following is *not* a Linux DHCP client?

    A. `pump`

    B. `dhcpcd`

    C. `dhcpd`

    D. `dhclient`

5. Which of the following types of information is returned by typing **`ifconfig eth0`**? (Select all that apply.)

    A. The names of programs that are using `eth0`

    B. The IP address assigned to `eth0`

    C. The hardware address of `eth0`

    D. The hostname associated with `eth0`

6. Which of the following programs can be used to perform a DNS lookup?

    A. `host`

    B. `dnslookup`

    C. `pump`

    D. `ifconfig`

7. Which of the following entries are found in the /etc/hosts file?

    A. A list of hosts allowed to remotely access this one

    B. Mappings of IP addresses to hostnames

    C. A list of users allowed to remotely access this host

    D. Passwords for remote web administration

8. Which of the following commands should you use to add to host 192.168.0.10 a default gateway to 192.168.0.1?

    A. `route add default gw 192.168.0.10 192.168.0.1`

    B. `route add default gw 192.168.0.1`

    C. `route add 192.168.0.10 default 192.168.0.1`

    D. `route 192.168.0.10 gw 192.168.0.1`

9. Which of the following pieces of information are usually required to initiate a PPP connection over an analog telephone line? (Select all that apply.)

    A. The ISP's telephone number

    B. The client IP address

    C. An account name (username)

    D. A password

10. What is the purpose of PAP and CHAP?

    A. They're tools to manage multiple servers, thus reducing the memory load of running many server programs.

    B. They're methods of converting IP addresses to hostnames,) or vice versa, for applications.

    C. They're protocols for exchanging username and password data in a standardized way in a PPP connection.

    D. They're competing systems for managing printers on a Linux system, with PAP being the more common one today.

11. What tool might you use to simplify PPP configuration, letting the tool figure out how to enter your authentication information?

    A. `netstat`

    B. `ppp-off`

    C. `wvdial`

    D. `pppd`

12. Which of the following are potential benefits of running a server via a super server? (Select all that apply.)

    A. Reduced memory load when running many small servers

    B. Improved server response speed

    C. Increased security from the super server's options

    D. Better capacity to retain data between connections to the server

13. You have just finished editing and changing the `inetd.conf` file. Which of the following commands will cause some Linux distributions to read the changed file?

    A. `/etc/inetd restart`

    B. `/etc/bin/inetd restart`

    C. `/etc/sbin/inetd restart`

    D. `/etc/rc.d/init.d/inetd restart`

14. You've installed a new server that includes its own `xinetd` configuration file in `/etc/xinetd.d`. This new server is not responding, even after you restart your distribution's standard `xinetd`. Which of the following is one of the first things you should check when debugging this problem?

    A. Type **`inetd verify`** to review the super server configuration to be sure that the new server is properly configured.

    B. Look for a line for the server in `/etc/inetd.conf`. If it's present but begins with a hash mark (#), remove the hash mark.

    C. Use `nslookup` to verify that `xinetd` is listening on the port for the new server. If it's not, type **`xinetd listen`** to fix the problem.

    D. Look for a `disable = yes` line in the configuration file in `/etc/xinetd.d`. If you find it, change it to read `disable = no`.

15. You need to add a printer definition to a stand-alone workstation running LPRng. Which file should you edit to add the printer?

    A. `/etc/cups/printers.conf`

    B. `/etc/printcap`

    C. `/etc/cups/cupsd.conf`

    D. `/etc/rc.d/init.d/lpd`

16. Which of the following describes the function of a smart filter?

    A. It detects the type of a file and passes it through programs to make it printable on a given model of printer.

    B. It detects information in print jobs that might be confidential as a measure against industrial espionage.

    C. It sends e-mail to the person who submitted the print job, obviating the need to wait around the printer for a printout.

    D. It detects and deletes prank print jobs that are likely to have been created by miscreants trying to waste your paper and ink.

17. What information about print jobs does the `lpq` command display? (Select all that apply.)

    A. The name of the application that submitted the job

    B. A numerical job ID that can be used to manipulate the job

    C. The amount of ink or toner left in the printer

    D. The username of the person who submitted the job

18. You've submitted several print jobs, but you've just realized that you mistakenly submitted a huge document that you didn't want to print. Assuming you can identify which job this was, that it's not yet printing, and that its job ID number is 749, what command would you type to delete it from the `okidata` print queue?

    A. The answer depends on whether you're using BSD LPD, LPRng, or CUPS.

    B. `lpdel -Pokidata 749`

    C. `lprm -Pokidata 749`

    D. None of the above; the task is impossible.

19. Which of the following is generally true of Linux programs that print?

    A. They send data directly to the printer port.

    B. They produce PostScript output for printing.

    C. They include extensive collections of printer drivers.

    D. They can print only with the help of add-on commercial programs.

20. What tool might you use to print a four-page PostScript file on a single sheet of paper?

    A. PAM

    B. `mpage`

    C. 4Front

    D. `route`

Answers to Review Questions

1. A, B, D. Ethernet is currently the most common type of network hardware for local networks. Linux supports it very well, and Linux also includes support for Token Ring and Fibre Channel network hardware. DHCP is a protocol used to obtain a TCP/IP configuration over a TCP/IP network. It's not a type of network hardware, but it can be used over hardware that supports TCP/IP.

2. B. IP addresses consist of four 1-byte numbers (0–255). They're normally expressed in base 10 and separated by periods. 63.63.63.63 meets these criteria. 202.9.257.33 includes one value (257) that's not a 1-byte number. 107.29.5.3.2 includes five 1-byte numbers. 98.7.104.0/24 is a network address—the trailing /24 indicates that the final byte is a machine identifier, and the first 3 bytes specify the network.

3. C. The gateway computer is a router that transfers data between two or more network segments. As such, if a computer isn't configured to use a gateway, it won't be able to communicate beyond its local network segment. (If your DNS server is on a different network segment, name resolution via DNS won't work, although other types of name resolution, such as `/etc/hosts` file entries, will still work.)

4. C. Option C, `dhcpd`, is the Linux DHCP *server*. The others are all DHCP clients. Most distributions ship with just one or two of the DHCP clients.

5. B, C. When used to display information on an interface, `ifconfig` shows the hardware and IP addresses of the interface, the protocols (such as TCP/IP) bound to the interface, and statistics on transmitted and received packets. This command does *not* return information on programs using the interface or the hostname associated with the interface.

6. A. The `host` program is a commonly used program to perform a DNS lookup. There is no standard `dnslookup` program, although the `nslookup` program is a deprecated program for performing DNS lookups. `pump` is a DHCP client, while `ifconfig` is used for configuration of networking parameters and cards.

7. B. The `/etc/hosts` file holds mappings of IP addresses to hostnames, on a one-line-per-mapping basis. It does not list the users or other hosts allowed to remotely access this one or affect remote administration through a web browser.

8. B. To add a default gateway of 192.168.0.1, the command would be **route add default gw 192.168.0.1**. Specifying the IP address of the host system is not necessary and in fact will confuse the `route` command.

9. A, C, D. You need a telephone number to dial the call (although this is *not* needed for a PPPoE or PPPoA broadband connection). Most ISPs use a username and password to authenticate access. Although you can specify an IP address, this option is only used in specialized circumstances.

10. C. Option C correctly describes the purpose of PAP and CHAP. Option A describes the function of super servers, such as `inetd` or `xinetd`. Option B describes the purpose of DNS or the `/etc/hosts` file. Option D describes the function of the BSD LPD, LPRng, and CUPS daemons.

11. C. The `wvdial` program simplifies PPP configuration and use by providing a somewhat "brainier" front end to `pppd` than the standard scripts. The `netstat` command of option A is a general-purpose network diagnostic and configuration tool, but it's not useful in automating PPP configuration. The `ppp-off` utility of option B is used to terminate a PPP connection, not to simplify the creation of such a connection. Option D's `pppd` is used to initiate a PPP connection, but by itself it's not a very friendly or simple tool; it's generally used with the help of a dialing script (`ppp-on`), `wvdial`, or a GUI dialing program.

12. A, C. Super servers can reduce memory load and have the potential to improve security by applying extra security checks on incoming connections. Super servers do *not* improve response speed or improve servers' ability to retain data between connections; if anything, these are benefits of running the server directly, without the help of a super server.

13. D. The `inetd` SysV startup script is usually located in `/etc/rc.d/init.d`, `/etc/init.d`, or `/etc/rc.d`. After changing the `inetd.conf` file, you can run this startup script with the `restart` or `reload` option to tell the `inetd` super server to implement the changes you've made. The other options all refer to the `inetd` SysV startup script in locations in which it never resides.

14. D. Many servers that ship with `xinetd` configuration files include the `disable = yes` line in these files as a security precaution. This configuration prevents the server from being active until the user explicitly activates it by changing `yes` to `no` on this line. Option A describes a nonexistent option to `inetd`, which isn't the right super server for this question. Option B's `inetd.conf` file is also the wrong configuration file, although this would be a reasonable thing to check if the system were running `inetd` rather than `xinetd`. The `nslookup` program is used to translate hostnames to IP addresses and vice versa; it's not a useful diagnostic tool in this situation.

15. B. You can add or delete printers by editing the `/etc/printcap` file, which consists of printer definitions for BSD LPD or LPRng. The `/etc/cups/printers.conf` file holds printer definitions for CUPS, and although you can directly edit this file to add a printer, doing so is tricky. `/etc/cups/cupsd.conf` is the main CUPS configuration file, `/etc/rc.d/init.d/lpd` is the BSD LPD printing startup script.

16. A. The smart filter makes a print queue "smart" in that it can accept different file types (plain text, PostScript, graphics, etc.) and print them all correctly. It does not detect confidential information or prank print jobs. The `lpr` program in the BSD, LPD, and LPRng printing systems can be given a parameter to e-mail a user when the job finishes, but the smart filter doesn't do this.

17. B, D. The job ID and job owner are both displayed by `lpq`. Unless the application embeds its own name in the filename, that information won't be present. Most printers lack Linux utilities to query ink or toner status; certainly `lpq` can't do this.

18. C. The `lprm` command deletes a job from the print queue. It can take the `-P`*queue* option to specify the queue and a print job number or various other parameters to specify which jobs to delete. BSD LPD, LPRng, and CUPS all implement the `lprm` command, so you can use it with any of these systems. Option B presents the correct syntax but the wrong command name; there is no standard `lpdel` command.

19. B. PostScript is the de facto printing standard for Unix and Linux programs. Linux programs generally *do not* send data directly to the printer port; on a multi-tasking, multi-user system, this would produce chaos because of competing print jobs. Although a few programs include printer driver collections, most forgo this in favor of generating PostScript. Printing utilities come standard with Linux; add-on commercial utilities aren't required.

20. B. The mpage utility prints multiple input pages on a single output page, so it's ideally suited to the specified task. PAM is the Pluggable Authentication Modules, a tool for helping to authenticate users. 4Front is the name of a company that produces commercial sound drivers for Linux. The route command is used to display or configure a Linux routing table.

Chapter

10

Managing Servers

THE FOLLOWING LINUX PROFESSIONAL INSTITUTE OBJECTIVES ARE COVERED IN THIS CHAPTER:

- ✓ **1.113.2 Operate and perform basic configuration of sendmail (weight: 4)**
- ✓ **1.113.3 Operate and perform basic configuration of Apache (weight: 4)**
- ✓ **1.113.4 Properly manage the NFS, smb, and nmb daemons (weight: 4)**
- ✓ **1.113.5 Setup and configure basic DNS services (weight: 4)**
- ✓ **1.113.7 Set up secure shell (OpenSSH) (weight: 4)**

Many Linux systems function as network server computers. These computers run one or more server programs as their primary reason for being. Even many computers that don't fit this description often run one or more server programs—say, to enable a workstation's primary user to log in remotely or to share files. For these reasons, this chapter covers six major Linux server programs and protocols: the *sendmail* mail server, the *Apache* web server, the *Network File System (NFS)* file server, the *Samba* file server, the *Berkeley Internet Name Domain (BIND)* name server, and the *Secure Shell (SSH)* protocol (and specifically the OpenSSH server).

> **NOTE** All of the servers described in this chapter are major Linux servers with many options and quirks. Entire books are devoted to each of them. Thus, this chapter can cover only the basics of getting these servers up and running and describe a few common options. If you need to know more, you must consult additional documentation, be it the documentation that ships with the server, web pages on the server, or books on the server. I provide pointers to additional documentation in each server's section.

Configuring Sendmail

E-mail is a very important part of networking. Many organizations rely on e-mail for day-to-day operations; without it, they'd grind to a halt. Naturally, Linux provides e-mail servers, which communicate with end-user systems and with other e-mail servers to manage your e-mail needs. Even workstations sometimes run e-mail servers in order to manage the e-mail that's generated locally. Thus, understanding at least the basics of how to configure an e-mail server will help you keep a Linux system running and may be extremely important if you must ever dabble in maintaining an organization's primary mail server system. This knowledge begins with a basic understanding of e-mail in general. You can then apply this knowledge to setting some basic mail server options. In addition to configuring the server, you should know a bit about how to manage the mail queue. Finally, an understanding of some e-mail security issues will help you avoid potentially disruptive configuration blunders.

> **NOTE** These sections focus on the sendmail mail server (`http://www.sendmail.org`), which is the most popular mail server on the Internet. Several Linux distributions now favor other mail servers, though, such as Postfix (`http://www.postfix.org`) or Exim (`http://www.exim.org`). Although the basic principles for all of these servers are identical, configuration details differ greatly. To learn more, consult the server's documentation or a book, such as Craig Hunt's *Linux Sendmail Administration* (Sybex, 2001).

E-Mail Basics

Several protocols exist to manage e-mail. The most common of these is the *Simple Mail Transfer Protocol (SMTP),* which is designed as a push mail protocol, meaning that the sending system initiates the transfer. This design is good for sending data, so SMTP is used through most of a mail delivery system. The final stage, though, often employs a pull mail protocol, such as the Post Office Protocol (POP) or the Internet Message Access Protocol (IMAP). With these protocols, the receiving system initiates the transfer. This is useful when the receiving system is an end user's workstation, which might not be powered on at all times or able to receive incoming connections.

SMTP was designed to enable a message to be relayed through an arbitrary number of computers. For instance, an end user might compose a message, which is sent to the local SMTP server. This server looks up a recipient system using the *Domain Name System (DNS)* and sends the message to that system. This system might use its own internal routing table to redirect the message to another local system, from which the message might be read, either directly or via a POP or IMAP server. This arrangement is illustrated in Figure 10.1. Bear in mind that the number of links in this chain is variable and depends on how each system is configured. In the simplest case, local e-mail stays on just one system. In theory, an arbitrarily large number of computers can be involved in an e-mail exchange, although in practice it's rare to see e-mail pass through more than half a dozen systems.

At each step in a relay chain, e-mail is altered. Most importantly, each server adds a *header* to the e-mail, which is a line that provides information about the message. In particular, mail servers add `Received:` headers to document the path the mail has taken. In theory, this enables you to trace the e-mail back to its source. Unfortunately, spammers and other e-mail abusers have learned to forge e-mail headers, which greatly complicates such analysis.

FIGURE 10.1   E-mail typically traverses several links between sender and recipient.
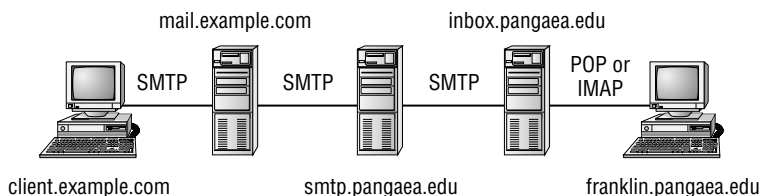
Because an SMTP server can function as both a server (receiving mail from other systems) and a client (sending mail to other systems), you must deal with both sides of the configuration equation, as described next, in "Setting Sendmail Options for Your System." Sometimes a system will never function in one role or the other, which can simplify matters—but you must then be careful not to accidentally configure the system incorrectly. In particular, *open relay* configurations, in which a mail server relays mail from anybody, should be avoided. This and other security implications of running an SMTP server are covered in "Sendmail Security Considerations."

Setting Sendmail Options for Your System

Linux distributions that use sendmail typically ship with configuration files that enable the server to function in a minimal way. Unfortunately, this may not include the ability to send mail to outside systems. To make changes, you must first understand the sendmail configuration files, which are complex. You can then set options to adjust your hostname, adjust from whom the server will accept mail, configure an outgoing relay, and make appropriate name substitutions on selected messages.

Sendmail Configuration Files

Sendmail uses a number of configuration files, each with its own unique purpose:

sendmail.cf   This file is the official primary sendmail configuration file. It includes information relating to relaying options, the server's hostname, and so on. Unfortunately, its syntax is very difficult for most people to understand, so in practice, it's usually created from another file that's easier to edit. You can usually find sendmail.cf in /etc/mail.

> **NOTE**   If your system lacks a sendmail.cf file, it's possible that your computer is running Postfix, Exim, or some other mail server program, or your system might not be running any mail server at all.

The m4 file   The file that's used to generate the sendmail.cf file is called an m4 file, after the m4 program that does the translation. This file's name varies from one system to another, but it usually ends in .mc. Examples include Red Hat's /etc/mail/sendmail.mc and Slackware's /usr/share/sendmail/cf/cf/linux.smtp.mc. This file is sometimes not installed with the main sendmail package; you may need to install it from its own package, such as sendmail-cf.

aliases   The /etc/aliases file (which sometimes appears in /etc/mail instead of /etc) holds username translations, as described in "Redirecting Mail." Sendmail actually reads a binary version of this file, aliases.mc, which is generated via the newaliases program, as described later.

access   The access file (which is usually stored in /etc or /etc/mail) controls mail relaying, as described later, in "Sendmail Security Considerations."

local-host-names   This file contains a list of hostnames that sendmail will treat as local. That is, if the server receives mail addressed to a user at one of these names, the server will attempt to deliver that mail to a local user of that name. If this file isn't used (its use can be disabled in sendmail.cf), or if it's empty, sendmail accepts mail addressed only to its own name.

If you check the /etc/mail directory, you'll see several additional configuration files. These files are important for more advanced configurations, but for the basic configuration tasks described in this chapter, you can ignore them.

To configure sendmail, you'll edit an m4 file and then convert it to a sendmail.cf file with m4:

```
# m4 < myconfig.mc > sendmail.cf
```

> **WARNING**   If you issue this command in the same directory in which the original sendmail.cf file resides, the command copies over the existing sendmail.cf file. For added safety, back up that file first. You can then restore it from the backup if something goes wrong.

Generally speaking, you'll want to start with the m4 file provided by your distribution but rename it to something appropriate. After you make a change and create a new sendmail.cf file, you must pass the reload or restart parameter to the sendmail SysV startup script, as described in Chapter 6, "The Boot Process and Scripts":

```
# /etc/init.d/sendmail reload
```

You should be aware that the m4 file specifies comments in an unusual way: The string dnl stands for a comment. Thus, if a line begins with dnl, it's a comment line. If you want to use that line in your configuration, remove the dnl characters.

Hostname Options

All mail servers must know their own hostnames. This information is used in a variety of ways, such as when initiating communications with other servers and as identification to be added to usernames for the benefit of clients that don't provide this detail. Ordinarily, sendmail retrieves its hostname from system calls designed for this purpose. (In most cases, these calls deliver the name that was in /etc/HOSTNAME or /etc/hostname at boot time.) Sometimes, though, you might want to override this hostname. For instance, your mail server might be known by multiple names and you want to set one for the system and another for sendmail alone. Sometimes you want outgoing e-mail to omit the machine name, as in ben@pangaea.edu rather than ben@franklin.pangaea.edu. To make such a change, look for lines like the following in the m4 file:

```
MASQUERADE_AS(`pangaea.edu')
FEATURE(masquerade_envelope)
```

> **NOTE**
>
> The MASQUERADE_AS line includes two types of single-quote marks. The first one (`` ` ``) is a backtick, which appears to the left of the 1 key on most keyboards. The second (') is an ordinary single quote, which is to the left of the Enter key on most keyboards. Be sure to get these right or your configuration won't work!

If you don't see such lines in your file, add them. You can then change the MASQUERADE_AS address to whatever you like. This change will add the specified hostname to any addresses in the From: header of outgoing mail that lack this information. The FEATURE(masquerade_ envelope) line takes things further, substituting the specified address for those that users specify in their mail clients.

Accepting Mail

Recently, sendmail configurations have begun to include a line like the following in their m4 files:

```
DAEMON_OPTIONS(`Port=SMTP,Addr=127.0.0.1, Name=MTA')dnl
```

This line has the effect of limiting sendmail to listening on the 127.0.0.1 (localhost) address. The result is that the server is available to local programs but not to other programs. This is a desirable configuration for most workstations and even non-mail servers. If you want sendmail to accept incoming mail or to function as a mail relay, though, you should remove this line. Ideally, you should comment it out by adding dnl to the start of the line. You can then rebuild your sendmail.cf file and have the server reload its configuration.

If you want sendmail to accept mail addressed to users of a whole domain or to a hostname other than the one the computer has, you should add those domain names or hostnames to the /etc/mail/local-host-names file. You should also ensure that the following line is present in your m4 file:

```
FEATURE(use_cw_file)
```

Outgoing Relay Configuration

Many Linux mail servers, particularly on workstations and small networks, should be configured to use an outgoing mail relay, aka a *smart relay*. To do so, enable outgoing relaying in your m4 file:

```
FEATURE(`nullclient', `relay.mail.server')
```

Change *relay.mail.server* to the hostname of the computer that should relay the mail. You should also remove or comment out two other lines, if they're present:

```
MAILER(local)dnl
MAILER(smtp)dnl
```

Another way to make similar changes is to use the SMART_HOST feature:

```
define(`SMART_HOST', `relay.mail.server')
```

The difference between the two is that the nullclient configuration is intended for very simple configurations that contain few other options. A workstation that should send *all* its e-mail via the outgoing relay might use this option. SMART_HOST, by contrast, can be used in more complex configurations in which some mail remains local and some is relayed.

When you've made these changes, rebuilt sendmail.cf, and restarted or reloaded sendmail, the server should send all non-local mail to the specified mail server. This is handy if your ISP blocks direct outgoing SMTP connections (as many home-oriented ISPs do as an anti-spam measure) or if you want your organization's mail to be filtered through a single powerful mail server system.

> **WARNING** In addition to sending its mail to a relay system, Linux running sendmail can function as a relay system. In most cases, this configuration is extremely undesirable. This topic is covered in more detail shortly, in "Sendmail Security Considerations."

Redirecting Mail

E-mail aliases enable one address to stand in for another one. For instance, all mail servers are supposed to maintain an account called postmaster. E-mail to this account should be read by somebody who's responsible for maintaining the system. One way to do this is to set up an alias linking the postmaster name to the name of a real account. You can do this by editing the aliases file, which usually resides in /etc or sometimes in /etc/mail.

The aliases file format is fairly straightforward. Comment lines begin with hash marks (#) and other lines take the following form:

```
name: addr1[,addr2[,...]]
```

The *name* that leads the line is a local name, such as postmaster. Each address (*addr1*, *addr2*, and so on) can be a local account name to which the messages are forwarded, the name of a local file into which messages are stored (denoted by a leading slash), a command through which messages are piped (denoted by a leading vertical bar character), the name of a file whose contents are treated as a series of addresses (denoted by a leading :include: string), or a full e-mail address (such as fred@example.com).

A typical default configuration includes a few useful aliases for accounts such as postmaster. Most such configurations map most of these aliases to root. Reading mail as root is inadvisable, though—doing so increases the odds of a security breach or other problem because of a typo or bug in the mail reader. Thus, you may want to set up an alias line like the following:

```
root: yourusername
```

This redirects all of root's mail, including mail directed to root via another alias, to *yourusername*. Once you make this, or any other, change to the aliases file, you must rebuild its binary counterpart, aliases.db. To do so, use the newaliases command:

newaliases

Another approach to redirecting mail is to do so on the user level. In particular, you can edit the ~/.forward file in a user's home directory to have mail for that user sent to another address. Specifically, the ~/.forward file should contain the new address—either a username on the current computer or an entire e-mail address on another computer. This approach has the advantage that it can be employed by individual users—say, to consolidate e-mail from multiple systems into one account without bothering system administrators. A drawback is that it can't be used to set up aliases for nonexistent accounts or for accounts that lack home directories. The ~/.forward file could also be changed or deleted by the account owner, which might not be desirable if you want to enforce a forwarding rule that the user shouldn't be able to override.

Managing the Mail Queue

Ideally, mail sent through sendmail leaves the originating system and arrives at its destination quickly. Unfortunately, this doesn't always happen. The recipient system (or the entire link to the Internet) could be temporarily down, or the network connection could be slow and the message large, or the mail server might be processing a large mass of e-mail messages. For these reasons, sendmail maintains a mail queue, which is similar in principle to the print queue maintained by the Linux printing system. When sendmail accepts a message for delivery to another system, the server stores it in a file. Sendmail makes an initial attempt to deliver the mail as soon as it can—but if sendmail is processing a lot of mail, this might not be instantly. If sendmail is unable to deliver the message because it can't reach the remote system, the message stays in the queue and sendmail eventually tries again. After a while (typically a few days; this detail can be adjusted), sendmail bounces the message—that is, it returns the message to the sender with a notice about the failure. Some types of failures result in an immediate bounce, as well.

You can check on the mail queue with the mailq command. In most cases, simply typing **mailq** is all that's necessary. The program responds with a summary of the messages in the queue. (Consult mailq's man page for information on its options.) This information can be a useful diagnostic tool. On a system that uses sendmail solely for local mail delivery and as an outgoing mail tool for local programs, the mail queue should have few or no messages in it at any given time. If lots of messages reside in the queue, it could be a sign of misconfiguration or of network problems.

If you've sent a message and it's not gone out, and if you believe you've corrected a problem that was preventing it from being sent, you can force sendmail to try sending all the messages in the mail queue. To do so, type **sendmail -q**. Depending on the number and sizes of the messages in the queue and your network connection speed, your mail queue will probably clear within a few seconds. If mailq shows that one or more messages haven't cleared, the problem may be at the receiving end. Sendmail will periodically attempt to send the message again.

Sendmail Security Considerations

Like any server, sendmail is a potential security risk. Broadly speaking, this risk takes two forms:

Bugs   Bugs in sendmail could expose your system to danger. In theory, a bug might enable somebody to gain access to your system by sending an e-mail or by connecting to the SMTP port (25) via a Telnet client and typing SMTP commands to trigger the bug. This is one reason many Linux distributions today limit access to sendmail to the local computer only.

Misconfiguration   Poor configuration of sendmail can cause problems. Sendmail wasn't designed as a way to give login access to a system, so you're unlikely to give an intruder full access to your system by a sendmail configuration. Instead, the big risk is a configuration that will make your system a menace to the Internet. The most common misconfiguration of this nature is an open relay, which is a computer that will relay mail from any computer to any other computer. In the past, spammers made heavy use of open relays as a way to help hide their true identities, but spammers today have largely moved on to other techniques. Nonetheless, some spammers still abuse open relays, so you should ensure that your system isn't configured as one.

All modern Linux distributions configure themselves so that they do *not* relay mail by default—or at least, they configure themselves so that they only relay mail for a very limited set of computers. The most common type of relay configuration in sendmail uses the following control line (or one similar to it) in the `m4` file:

```
FEATURE(`access_db')
```

This line tells the system to read the `access` file (stored in `/etc` or `/etc/mail`) for information on who may relay messages. Listing 10.1 shows a typical `access` file. This example file enables relaying for the local computer (as `localhost.localdomain`, `localhost`, and `127.0.0.1`) and for the local network (172.25.98.0/24). Most default configurations omit this last option, which appears in Listing 10.1 only to show that it's possible. If your system shouldn't be relaying mail, the `access` file should lack any line like this one.

Listing 10.1: A Typical *access* File for Controlling Mail Relaying

```
# by default we allow relaying from localhost...
localhost.localdomain          RELAY
localhost                      RELAY
127.0.0.1                      RELAY
# Relay for the local network
172.25.98                      RELAY
```

Some other `m4` file relaying options are also available in sendmail. Most of these include the word `relay` in their names, such as `relay_entire_domain`. Unless you've learned a great deal about e-mail configuration and management, I recommend you avoid using such options. In fact, one of these options, `promiscuous_relay`, should *never* be used. It tells the computer to indiscriminately relay *all* mail. In the days before spam, such configurations were considered a courtesy. Today, they're madness!

Because open relays are such a threat, you should explicitly check for them. Review your `access` file and look for any option with the word relay in the `m4` configuration file. You can also use services that will connect to your mail server and attempt to relay mail as a way of testing your system, then give you a report of the results. One of these is hosted at `http://www.abuse.net/relay.html`. Be careful, though—it's conceivable that a spammer somewhere has set up such a service as a way of locating open relays to abuse, so you should only use a trusted service.

Configuring Apache

The World Wide Web (WWW or Web for short) is arguably the highest-profile part of the Internet. As such, it should come as no surprise that Linux provides servers to deliver Web content. The most popular of these, Apache (`http://httpd.apache.org`), is an extremely flexible web server. Fortunately, it's also fairly straightforward to configure for simple sites, although doing complex things with it requires far more information than I can present in this chapter. Consult Apache's own documentation or a book on the subject, such as Charles Aulds' *Linux Apache Web Server Administration, 2nd Edition* (Sybex, 2002) for more details.

To get started, though, you should understand a bit of how Apache works. You can then dig into the Apache configuration files to get the server working as it should. Finally, as with all servers, you should be aware of Apache's security implications so that you don't open the door to miscreants when you run Apache.

Apache Basics

A web server listens on a port (typically port 80) for connection requests and communicates with clients using the *Hypertext Transfer Protocol (HTTP),* hence the string `http://` that begins URLs for web pages. Web servers primarily deliver files to clients rather than receive files from them, although some web pages and server configurations permit users to upload files to the server, as well. In most cases, another protocol, such as the File Transfer Protocol (FTP), is used to upload the web pages themselves to the web server computer. If users have direct login access to the web server, though, web pages could be created on the web server system itself.

Most web pages use the Hypertext Markup Language (HTML), typically indicated by a `.htm` or `.html` filename extension. Web servers can deliver other types of files, though, and in fact most web pages refer to graphics files in various formats, which web browsers display automatically. This chapter doesn't describe the creation of HTML or other web documents, though, just the configuration of the web server that delivers them to clients.

Apache's configuration file takes on various names depending on the distribution. The most common are `httpd.conf`, `httpd2.conf`, `apache.conf`, and `apache2.conf`. (The names that include a 2 are used by Apache 2.0 and later.) Similarly, the name of the Apache executable can vary, but is usually `httpd`. The location of the Apache configuration file also varies from one system to another, but common places include `/etc/apache`, `/etc/apache2`, `/etc/httpd`, `/etc/httpd2`, or the `conf` subdirectory of any of these.

If you can't find your Apache configuration file, use your package manager to find it. For instance, if you installed Apache on an RPM-based system as apache, you could type **rpm -ql apache | grep conf**. This command lists all the files in the apache package and narrows the list to those that include the string conf. Chapters 2 and 1 describe package management and grep, respectively.

Whatever its name, the Apache configuration file consists of comments, denoted by hash marks (#), and directive lines that take the following form:

Directive Value

The *Directive* is simply an option name or command, such as LoadModule, Listen, or MaxClients. In most cases, the *Value* is a filename, number, or other single value that's assigned to the *Directive*. Sometimes the *Value* consists of multiple items; for instance, LoadModule takes a two-part directive that consists of a module name and a module filename. The parts of such a multi-part *Value* are separated by spaces.

Some option lines are grouped together in blocks, which are identified by beginning and ending lines in angle brackets (<>), with the name of the ending angle bracket line beginning with a slash (/):

```
<IfModule perchild.c>
NumServers          5
StartThreads        5
MaxThreadsPerChild  20
</IfModule>
```

Typically, these blocks apply only if a particular condition exists. For instance, the <IfModule perchild.c> block shown here applies only if the perchild.c module is available and loaded. (Modules are described shortly.)

Like most big servers, Apache is usually run via SysV startup scripts, which are described in Chapter 6. After making changes to an Apache configuration, you should pass the restart or reload option to the Apache startup script:

/etc/init.d/apache reload

Alternatively, on some systems you may use the apachectl program, which takes the same arguments as most SysV startup scripts—start, restart, reload, status, and so on. Not all distributions ship their Apache packages with the apachectl program, though, presumably because it's redundant with SysV startup scripts.

You do *not* need to restart or reload Apache after you change your web pages. Apache should begin delivering the new files as soon as they replace the old ones. Most browsers, though, cache web pages and so may not immediately show a change.

   Prior to version 2.0, Apache could be run from a super server. (Chapter 9, "Basic Networking," describes super server configuration.) Running Apache in this way could be handy if Apache is seldom accessed, but it slows down the server's responses to clients. With version 2.0, the ability to run from a super server was removed from Apache.

Setting Apache Options for Your System

Apache is an extremely complex server, so in the few pages devoted to it in this chapter, I can only provide information on a few Apache options. In most cases, an Apache package installed on a Linux system works in a minimal way as soon as it's run. That is, the server delivers web pages to clients when they contact the server. These configurations typically include a default web page to serve as a placeholder page until you store your own website's HTML files in the default file location. (You can adjust this location, as described shortly.) If you load your Apache configuration file into a text editor, you might want to search for, and possibly modify, several directives:

LoadModule   Apache supports modules, which are pieces of Apache support code stored in separate files. Using modules enables you to load just those modules you want, minimizing the memory impact of running Apache. As a general rule, you should leave the LoadModule directives in your Apache configuration file alone. You might find documentation indicating that you must enable or disable a particular module to achieve a particular effect, though. If so, follow the directions you find.

Include   An Include directive loads another file as if it were part of the main Apache configuration file. Some distributions break certain Apache options into separate files as a way of managing them.

User and **Group**   These options set the user and group that Apache uses to run. Typically, they're set to low-privilege values, perhaps associated specifically with Apache.

DocumentRoot   This directive sets the root of the document tree that Apache delivers. For instance, if DocumentRoot on www.example.com is set to /var/www/htdocs, and if a user enters http://www.example.com/products/mega2.htm as a URL, Apache returns the file /var/www/htdocs/products/mega2.htm (if it exists). Even if you don't want to change this location, you must know what it is so that you can place your own web pages in that location. With the exception of user files (described next), Apache will return only files within the DocumentRoot directory tree or files linked to this tree.

> **NOTE**   You may need to change the ownership or permissions on the DocumentRoot directory so that authorized individuals may add web pages to that location. If a particular group already has write privileges to the location, adding users to that group, as described in Chapter 8, "Administering the System," may do the trick.

UserDir   On a multi-user system, individuals can have their own web pages. These are typically accessed by preceding the username with a tilde (~) in the URL, as in `http://www.example.com/~sally/` to access `sally`'s home page. The `UserDir` directive tells Apache which directory to use as the root for these user directories. For instance, setting `UserDir public_html` causes Apache to look in a user's `public_html` subdirectory for user web pages. This directive relies on the `userdir_module` module, either compiled directly into Apache or loaded via a `LoadModule` directive.

BindAddress   This directive specifies the network interface to which the server listens. Setting it to an asterisk (*) causes Apache to bind to all interfaces, which is desirable on most true Apache servers. If you're running Apache for purely local purposes (say, to gain access to HTML documentation files on a workstation), setting `BindAddress` to `127.0.0.1` can improve security by preventing outsiders from accessing the server.

Listen   This directive tells Apache which port it should use. The default value of 80 is appropriate in most cases, but if you have cause to run Apache on a peculiar port, you can change this directive.

Apache supports many more options, but these are the ones you're most likely to want to adjust for a small Apache server used to serve content to the computer on which it runs or to a small network. If you want to run Apache on the Internet as a whole, you should research the server in much greater depth. You may need to use more advanced options to achieve particular effects or to improve security. This is particularly true if you want to use *Common Gateway Interface (CGI)* scripting features, which enable Apache to run programs at the request of clients. If improperly configured, CGI scripts can quickly lead to security breaches, but they're helpful and even necessary for some web server functions.

> CGI scripting relies on a module called `cgi_module`. Commenting out the `LoadModule` line that activates this module can reduce the risk of a security breach if you don't intend to use CGI scripting on your web server.

Apache Security Considerations

Web servers as a class are vulnerabilities because they're very high in visibility. This fact makes them a target for crackers who want to break in to deface a website, or perhaps to do other things. Web servers also necessarily give greater access to the host computer than do mail servers or certain other types of servers. Although limiting access to the `DocumentRoot` directory does reduce the risk, it doesn't eliminate the risk, particularly if the system is misconfigured with an unfortunate link between a file in this directory and a more sensitive system file. Finally, as noted earlier, CGI scripts can be a major risk factor.

Several of the options described earlier, in "Setting Apache Options for Your System," can help you keep your Apache server secure. Of particular import are the `User`, `Group`, and `BindAddress` directives. (The `BindAddress` directive can improve security on workstations that run Apache for local use only and on systems with multiple network interfaces, but isn't likely to be helpful for typical Apache servers that should be accessible to the outside world.)

You can also use non-Apache tools to help improve Apache security. In particular, you can use a firewall (described in Chapter 7, "Documentation and Security") to restrict what systems can access your Apache server. This technique is most likely to be helpful for internal Web servers—those that you run for the benefit of your local network's users rather than for the public at large.

Configuring an NFS Server

A mainstay of local network servers is the *file server,* which is a server that's intended to enable storage of files from the client. Many file servers can be made to look like local filesystems to clients. The result is that users can access their files from any client on the network and they can share files with other users. This can help save money by reducing the need for large hard drives on client computers. It also greatly simplifies backup—rather than back up user data files scattered across dozens of client systems, you can back up the files from a single server computer.

In the Unix and Linux worlds, the most common file server protocol is NFS. Before embarking on NFS configuration, you should understand its basic principles of operation. You can then configure an NFS server and an NFS client. As with all servers, you should be aware of the security implications of running an NFS server. Although NFS is simpler to configure than most of the servers covered in this chapter, its complexity is greater than I can describe here. For more information, consult a book on the subject, such as Erez Zadok's *Linux NFS and Automounter Administration* (Sybex, 2001).

> **NOTE** Another important file-sharing protocol is SMB/CIFS, which is described later, in "Configuring Samba."

NFS Basics

The Network File System (NFS) was developed by Sun as the file-sharing protocol for its Solaris OS and has been widely adopted among other Unix and Unix-like OSs. Because NFS was designed with Unix in mind, it's a good match to Unix (and hence Linux) requirements. In particular, NFS supports file ownership (both user and group) and permissions much as does Linux. NFS also supports hard and symbolic links and other special file types.

NFS was developed at a time when network security wasn't as complicated as it is today. Instead of requiring a password to authenticate users, NFS relies on a *trusted hosts* model of security: The server maintains a list of computers that are trusted to access files. If one of these trusted hosts accesses the server, the server grants more-or-less complete access to the files, relying on the client's own local security measure to restrict access appropriately according to the owner, group, and mode of the files. This isn't to say that NFS gives full access to all the files on the server computer, though; like Apache, NFS restricts access to specified directories (known as *exports*). NFS also provides some limited tools for restricting access on a per-user basis, as described shortly, in "Creating File Exports."

One potential problem with NFS is that, like Linux filesystems, NFS identifies all users and groups by number. The mapping from these user ID (UID) and group ID (GID) numbers to usernames and group names occurs on the client. This can work fine, but if two clients have different mappings, problems can ensue. For instance, suppose that two users, `ben` and `sally`, have accounts on two client systems, `client1` and `client2`. If `ben`'s UID on `client1` is 527 and his UID on `client2` is 528, `ben` may not be able to access files he created on `client1` when using `client2` and vice versa. Worse, if `sally`'s UIDs on `client1` and `client2` are 528 and 527, respectively, their ownership of files could end up being reversed, with files created by `ben` from `client1` appearing to be owned by `sally` when accessed from `client2`.

This potential for UID and GID problems means that if you use NFS, you should take whatever steps are necessary to ensure that UID and GID numbers are synchronized across clients. On a small network with a handful of clients, this can be done manually, by taking care when creating new user accounts. On a larger network, you might need to use a centralized login server system, such as the Network Information System (NIS) or a Lightweight Directory Access Protocol (LDAP) server. Such configurations are beyond the scope of this book, though.

Over its history, Linux has supported several different NFS servers. The latest crop of servers relies on support in the Linux kernel to optimize performance. This server, like earlier Linux NFS servers, is traditionally run via SysV startup scripts, as described in Chapter 6. The startup script name is usually `nfs` or `nfsserver`.

Creating File Exports

In Linux, NFS server configuration is handled through a file called `/etc/exports`. This file contains lines that begin with a directory that's to be shared followed by a list of hostnames or IP addresses that may access it, with their options in parentheses:

```
/home taurus(rw,async) littrow(ro)
/opt taurus(ro)
```

These examples share two directories: `/home` and `/opt`. Two computers (`taurus` and `littrow`) may access `/home`, but only `taurus` may access the `/opt` export. You can also specify hosts using wildcards (such as `*.luna.edu`, which matches all systems in the `luna.edu` domain), using IP addresses (as in `192.168.1.24`), using an IP address/netmask pair (as in `192.168.1.0/24`), or as an NIS netgroup if your network supports NIS (as in `@agroup`). The options in parentheses specify security restrictions and other server features, such as the following:

secure and **insecure**   The default operation is `secure`, which denies access requests from port numbers above 1024. On Unix and Linux systems, only `root` may initiate connections using these secure ports, so this is a security feature—but not a very effective one in today's environment. You can disable this protection by specifying the `insecure` option.

ro and **rw**   These options specify read-only or read/write access to the export, respectively. The default has changed over time; most current servers use `ro` as the default, but some older ones used `rw`. I recommend you set this option explicitly to avoid any potential for confusion.

sync and async   Ordinarily, an NFS server is supposed to complete certain operations by writing data to disk before responding to the client. This configuration is denoted by the sync option. You can improve performance by specifying async instead, which allows the server to respond to clients before the disk operation is completed. The drawback is that data loss is more likely in the event of a server crash.

root_squash and no_root_squash   By default, NFS treats accesses from the client's root user (UID 0) as if they were coming from a local anonymous user, as set by the anonuid and anongid options, described shortly. This *squashing* is a security measure that can be overridden by specifying the no_root_squash option. Ordinarily, you shouldn't use this option because it opens the door to abuses should a client system be compromised. It might be necessary in some cases, though.

all_squash and no_all_squash   Just as root access can be squashed, access by other users can be squashed. Specifying all_squash does so, giving access by all users a low privilege status. This configuration can be useful if you want to share a public directory to which users should be unable to write.

anonuid and anongid   You can set the UID and GID of the anonymous account used for squashing operations.

In most cases, just a few options are necessary for most exports. You should always specify rw or ro, simply to avoid the possibility of confusion, since the default for this option has changed in the past. The async option can be a useful one if you want the best possible performance, but it increases the risk of losing data in the event of a server crash. As an added security measure, you might want to use the all_squash option on publicly accessible NFS exports—say, an export that holds clip art that should be accessible to everybody on your network.

You can specify different options for different computers. For instance, you might give one system full read/write access to the server but deliver read-only access to others. This configuration enables one person or a group of people (the users of the read/write systems) to change the contents of an export—for instance, to update that clip art collection. After making changes to /etc/exports, you must tell the server to implement these changes. You can do this by typing **exportfs -ra** as root to do the trick. Many distributions' SysV startup scripts for NFS also do this when you pass the reload option.

Mounting NFS Exports

NFS exports may be mounted in either of two ways:

- You can mount the share using the mount command, much as you can mount a filesystem stored on a local partition.
- You can add an entry to the /etc/fstab file, much as you can create an entry for a filesystem stored on a local partition.

Chapter 4, "Managing Files and Filesystems," describes the mount command and /etc/fstab file in detail, so consult it for general information on both of these topics. In either event, mounting an NFS export works much like mounting a local filesystem. The main difference is that you must

specify the NFS server name and the export name using the format *server*:*/export* rather than specify a local device filename. For instance, to mount the /opt export at /share/opt from the system challenger.luna.edu, you might type the following command:

```
# mount -t nfs challenger.luna.edu:/opt /share/opt
```

This example specifies the filesystem type code of nfs, which is used for NFS exports; however, in most cases Linux can figure this detail out, so you can omit it. Depending on the client's domain name and domain search options, you may also be able to omit the domain portion of the server's name, as in changing challenger.luna.edu to challenger.

To have Linux automatically mount an NFS export at boot time, add an entry to /etc/fstab:

```
challenger:/opt  /share/opt  nfs  defaults  0 0
```

You use the same format for the export specification in /etc/fstab as you do with the mount command. This example shortens the hostname portion. This example also specifies defaults as the mount options, but you can tweak how the client handles the export by specifying various options, including:

rsize=*value* and wsize=*value*   These options specify the size of the read and write blocks transferred with the server. The default for value in both cases is 4096, but increasing the value to 8192 may improve performance.

hard   By default, Linux's NFS client implementation returns no error code when the server doesn't respond. This causes programs to hang when a problem occurs but to continue working once the server comes back online. To reiterate this default, specify the hard option.

soft   This option changes Linux's error procedure. If the server doesn't respond, the kernel returns an error code to the user program, which typically causes the program to deliver an error message to the user. This option can cause spurious error messages in the event of a minor network glitch but might be preferable to a prolonged program hang in some cases.

udp and tcp   Ordinarily, NFS operates using the User Datagram Protocol (UDP) low-level protocol. You can reiterate this by specifying the udp option. If you prefer, you can specify tcp, which causes NFS to use the Transmission Control Protocol (TCP) instead. TCP includes extra error checking and correction systems, which can make for more reliable connections; but these features also add overhead, which can degrade performance slightly. Many older NFS servers only support UDP operation.

NFS Security Considerations

NFS is a useful protocol, but like all network protocols, it's not without its risks. One of the more unusual security features of NFS is its trusted-hosts security model. Unlike most servers that provide the level of access that NFS provides, NFS doesn't use passwords or any other authentication tool to verify the client's identity. This makes the specification of authorized clients in /etc/exports extremely critical. If you authorize inappropriate clients in this file, you'll be risking unauthorized access to NFS-served files. To minimize this risk, you should specify hosts individually whenever

possible rather than by using NIS netgroups, wildcards, or IP address blocks. Furthermore, using IP addresses is generally preferable to using hostnames. If you use hostnames, your NFS server's security becomes partially dependent upon the security of the DNS server. If the DNS server is compromised, the intruder could alter its entries to gain access to the NFS server. Of course, all of these precautions amount to nothing if an authorized client is compromised or if an intruder gains physical access to your network wiring. Because wireless networks can be compromised fairly easily, NFS is rather risky to use on such networks. Be aware that NFS doesn't encrypt the data it transfers, so if an intruder can monitor your network activity, all the files stored on an NFS server can become visible as they are used.

NFS is best employed for storing user data files and executable program files that are needed on many computers on a network, thus simplifying backup and reducing disk space requirements. To these ends, you might export directories such as `/home`, `/usr`, and `/opt`. You should *not* generally export sensitive system-specific directories, such as `/etc`. In the event of a server bug or misconfiguration, exporting such a directory could enable an intruder to change critical system files such as `/etc/passwd`.

Configuring Samba

The Server Message Block/Common Internet File System (SMB/CIFS) protocol serves a role in the Windows world similar to that played by NFS in the Unix world. Fortunately, Linux can interact with Windows systems using SMB/CIFS; the tool enabling this interaction is known as Samba (`http://www.samba.org`). In fact, Samba works so well that many network administrators have replaced Windows NT/200*x* file servers with Linux systems running Samba. Its security implications are different from those of NFS—in fact, for this reason you might consider using SMB/CIFS rather than NFS even on a Linux-only network!

> **NOTE**   Samba is a server that's comparable in complexity to Apache or sendmail. As such, I can only provide a brief overview in this chapter. For more details, consult the documentation on the Samba website or a book on the topic, such as my *Linux Samba Server Administration* (Sybex, 2000) or my *Definitive Guide to Samba 3* (Apress, 2004).

Samba Basics

SMB/CIFS is primarily a DOS and Windows file-sharing protocol, although a few features are geared toward other OSs. This orientation provides certain challenges for a Linux SMB/CIFS server. For instance, Samba must be able to manage DOS and Windows filename conventions, which differ from those of Unix and Linux. For the most part, Samba handles these differences transparently, but you can tweak various configuration options to change how Samba manages such features.

The SMB/CIFS security model is quite different from that of NFS. Like most servers that provide extensive access to the system, SMB/CIFS works around a username and a password. (Some configurations can omit these, but for the most part, they're used.) Samba can use either encrypted or unencrypted passwords. Unfortunately, the encryption methods used by SMB/CIFS are incompatible with the standard form of Linux's encrypted password database, which means that if you use encrypted passwords, Samba must maintain its own password database independent of the standard Linux password database. Keeping the two synchronized can be a challenge, but Samba provides tools to help. Alternatively, you can use advanced Pluggable Authentication Module (PAM) features to enable Linux to use the Samba password database instead of the regular Linux passwords.

Unlike NFS, which is strictly a file-sharing tool, Samba provides for both file and printer sharing, as well as several ancillary tools. File and printer shares can be configured in nearly identical ways in Samba.

SMB/CIFS was originally designed to be used over a non-TCP/IP networking protocol known as NetBEUI. NetBEUI was designed as a purely local networking tool, though, so as the Internet (and the TCP/IP protocol upon which it's built) grew in popularity, using SMB/CIFS over TCP/IP became the norm. Samba uses SMB/CIFS over TCP/IP. Nonetheless, a few NetBEUI-related quirks exist in the SMB/CIFS way of looking at things. One of these is in computer naming; SMB/CIFS uses its own naming system (NetBIOS names) that's independent of the TCP/IP naming system. In most cases it's best to try to synchronize the two naming schemes. If you don't, you can run into some odd problems when you try to use a TCP/IP hostname in a tool that expects NetBIOS names or vice versa.

The Samba server suite consists of several individual server programs and an assortment of configuration files and support utilities. The two main server programs are `smbd`, which handles the main file- and printer-sharing duties, and `nmbd`, which handles NetBIOS name resolution and various other ancillary functions. In addition, the `swat` server provides a Web-accessible GUI configuration tool. The `smbd` and `nmbd` servers are usually run via SysV startup scripts, although they can be run from a super server. The `swat` server is almost always run via a super server.

Using Samba as a Server

The main role for Samba is as a server. Configuring a Samba server requires setting two main types of options: global options and options for individual shares (file shares and printer shares). Ultimately, these options are set in the main Samba configuration file, `smb.conf`. (This file usually resides in `/etc/samba`, although some older distributions used `/etc/samba.d` or some other location.) If you like, though, you can use the Samba Web Administration Tool (SWAT) to configure Samba using a point-and-click interface.

> The man page for `smb.conf` is unusually complete. Consult it for information on `smb.conf` options.

Setting Global Samba Options

The smb.conf file is split into multiple sections, each of which begins with a section name in square brackets, such as [global], [homes], or [bigshare]. Most sections define file or printer shares, as described shortly, in "Creating File Shares" and "Creating Printer Shares." The [global] section, though, sets default values for all file and printer shares as well as options that make sense only to the server as a whole.

Within each section, Samba parameters take the following form:

```
parameter = Value
```

The *parameter* is an option name and the *Value* is the value that should be assigned to the parameter. Comments are indicated by hash marks (#) or semicolons (;).

Most distributions ship with smb.conf files that serve as reasonable starting points for Samba configuration. Nonetheless, chances are you'll need to tweak at least a few global options to get a working system:

workgroup   The workgroup parameter sets the name of the NetBIOS workgroup or domain to which the system belongs. This parameter must be set to the correct value for the Samba server to be visible to most Windows clients. If you don't know what value to use, consult your network administrator. If you're setting up a new network, you can use just about any name you like. Note that a NetBIOS domain, if you use one, is unrelated to an Internet domain. You can certainly use your Internet domain name as a NetBIOS workgroup or domain name if you're setting up a new network, but you don't need to do so.

netbios name   By default, Samba uses your TCP/IP hostname (minus the domain portion) as its NetBIOS name. You can override this behavior by specifying a name with the netbios name parameter.

security   The security parameter tells Samba how to authenticate users. For a simple configuration, setting security = User makes sense. This corresponds to normal authentication using the local account database (either Samba's or Linux's, depending on other options). More advanced configurations might use Server or Domain, but such configurations require that your network have a domain controller and are beyond the scope of this chapter's Samba coverage. Setting security = Share eliminates the need to maintain individual user accounts, but you must then link each share to a set of local Linux usernames. The rules for doing so are complex, but if you must try this option, set one or more account names with the users parameter.

encrypt passwords   This parameter tells Samba whether or not to support encrypted passwords. In most cases, setting this parameter to Yes is appropriate because all recent versions of Windows expect to use encrypted passwords. Unfortunately, doing this means that you must maintain a local Samba password database. You can do so with the smbpasswd utility, which works similarly to the regular Linux passwd utility. The default for this parameter changed with Samba 3.0, from No to Yes. Thus, I recommend you set it explicitly to avoid confusion.

local master and **domain master**   These options both relate to master browser configuration, which is a NetBIOS tool for locating servers from clients. Samba can serve as a master browser, but proper configuration is an advanced topic, and incorrect configuration can cause

network disruptions. Thus, you should ensure that both of these options are set to No unless you research this topic more and want to set up Linux as a master browser.

wins server   This option specifies the IP address of the Windows Internet Naming Service (WINS) server, which is the NetBIOS name service tool. It's important for some Samba client tools, and setting this option lets these tools operate as WINS clients.

name resolve order   This option specifies the order in which various tools should be used to resolve names. Possible options are wins (for WINS), bcast (for NetBIOS-style broadcast name resolution), lmhosts (for the /etc/samba/lmhosts file), and host (for Linux's standard TCP/IP name resolution). You can specify from one to all four of these, separated by spaces, as in name resolve order = wins bcast host. This option is most important for client functions.

Many other global Samba options exist and may be important for certain configurations, but are beyond the scope of this chapter. Consult additional Samba documentation for details.

Creating File Shares

File shares are fairly simple to configure. A basic share looks like this:

```
[sample]
```

This single line creates a share called SAMPLE. This share isn't likely to be very useful, though, because it uses defaults for many important parameters you're likely to want to customize. In particular, this default share provides read-only access to the /tmp directory. A more useful share looks something like this:

```
[useful]
   comment = User Programs
   read only = Yes
   write list = ecernan
   path = /samba/userprogs
```

The comment parameter provides a comment field that's visible in Windows file browsers; it can be used to help identify the purpose of a share. The read only = Yes line reiterates the default state of a share as being read-only; it's not strictly necessary, but specifying it makes your intent in creating the share plain. The write list parameter specifies users who should be able to write to the share even though it's read-only. The intent in this case is that ecernan should be able to maintain the files in the share. Finally, and in some ways most importantly, the path parameter specifies the directory that corresponds to the share. In this case, users who access the USEFUL share will be reading from (or possibly writing to, for ecernan) the /samba/userprogs directory.

> **NOTE**   Linux's normal file-access controls apply to Samba access. For instance, in the preceding share, ecernan must be able to write to /samba/userprogs (or its sub-directories) in order to maintain the USEFUL share. Furthermore, if a file in that directory has permissions that prevent certain users from reading it, those users will be unable to read the file via Samba.

Many Samba parameters have synonyms, and a few have antonyms. For instance, read only = Yes, writeable = No, and writable = No all have identical effects. Similarly, the path and directory parameters have the same meaning.

One file share has special meaning: the [homes] share. This share normally lacks a path parameter because that parameter is set to the user's home directory. This share may also be accessed by each user's username. For instance, if ecernan accesses the ECERNAN share, Samba uses /home/ecernan (or whatever directory is specified in /etc/passwd). Most sample smb.conf files include a working [homes] share. If you want users to be able to store files in their home directories, this configuration may not require any changes. If your Samba server isn't to be used for storing personal files, though, you might want to remove that share.

> You can quickly deactivate a share by setting the available = No parameter in the share. This deactivates the share, but you can easily make the share active again by removing this line or changing No to Yes.

EXERCISE 10.1

Create a Samba File Share

In this exercise, you'll create a new Samba file share, which can be accessed by Windows, Linux, and other systems on your local network. This share will hold files that should be readable by all local Samba users. Perhaps this will be a share for your organization's document templates. This exercise assumes that Samba is already up and running on your system. To create a new share, follow these steps:

1. Log into the Linux system as a normal user.

2. Launch an xterm from the desktop environment's menu system, if you used a GUI login method.

3. Acquire root privileges. You can do this by typing **su** in an xterm, by selecting Session ➢ New Root Console from a Konsole, or by using sudo (if it's configured) to run the commands in the following steps.

4. Create a directory that you'll share, such as /home/samba/templates. For instance, you might type **mkdir -p /home/samba/templates**.

5. Set permissions on the directory you've just created so that all users can read from it but only the owner can write to it. Typing **chmod 0755 /home/samba/templates** will do this.

6. Assign ownership of the new directory to an appropriate account. For instance, if ecernan will be maintaining the files in this directory, type **chown ecernan /home/samba/templates**.

7. Type `ls -ld /home/samba/templates` to verify that the ownership and permissions are set correctly on this directory.

8. Load the /etc/samba/smb.conf file into your favorite text editor.

9. Review the existing shares. Pay particular attention to the share names; you don't want to use an existing name for your new share.

10. At the end of the file, enter lines to create a new file share:

```
[templates]
   comment = Templates share
   path = /home/samba/templates
   read only = No
```

11. Save the **smb.conf** file and exit from the editor.

12. Type **/etc/init.d/smb reload** to force Samba to reload its configuration file. (Some systems store the Samba SysV startup script in another directory or call it something else, so you may need to modify this command.)

At this point, the new share should be accessible. You can test it with smbclient, as in **smbclient //localhost/templates**. (Type this command as an ordinary user with a Samba account, not as root.) The result is an FTP-like exchange, enabling you to copy files back and forth, rename files, delete files, and so on.

Creating Printer Shares

Printer shares are just like file shares in most respects. The most important difference is that printer shares include the `printable = Yes` parameter (or its synonym, `print ok = Yes`). If this parameter is present, Samba looks for a local printer queue with the same name as the share definition. For instance, consider the following share:

```
[okidata]
   comment = Okidata printer
   printable = Yes
   path = /samba/okidata
```

This share appears as a printer share on Windows clients. When a client submits a print job to the share, Samba looks for a local printer queue called okidata and tries to send the print job to it. If you want the printer to appear under a different name to Samba clients than the local queue name, you can use the `printer` or `printer name` parameter to specify the local queue name. For instance, `printer = oki` causes Samba to submit the print job to the oki queue, no matter what the share name.

> **NOTE**   Printer shares don't need to be explicitly configured as read/write shares; this feature is implicit with their status as printer shares.

Printer shares have the equivalent to the [homes] file share: the [printers] share. If this share is present (as it is in many sample smb.conf files), it creates a printer share for every locally defined printer (in /etc/printcap or as defined in CUPS, depending on your local printing system). In many cases, this share is all you need to share your printers. If you explicitly create a share with the same name as one that would be created by [printers], your explicitly created share takes precedence. You can use this fact to override an automatically created share, say to specify unusual options.

As described in Chapter 9, Linux treats printers as PostScript devices. Thus, you should normally install Samba-shared printers in Windows as if they were PostScript printers. Typically, Apple LaserWriter drivers work well for monochrome printers and QMS magicolor drivers work well for color printers. If you create a raw queue under Linux, though, the Linux printing system won't force the print job to be treated as PostScript. You can then install the printer manufacturer's drivers under Windows. Both approaches have their advantages. The PostScript road is typically easier to configure and can produce lighter network loads but may block access to advanced printer features. The raw queue approach is more likely to give full access to all printer features but is likely to generate more network traffic.

Using SWAT

Although you can configure Samba via its smb.conf file, you might prefer to use a GUI interface. Some distributions provide such interfaces as part of their system administration tools, such as SuSE's YaST2. With all distributions, though, you can use SWAT, which ships with Samba. SWAT uses HTTP, so you can access SWAT using a web browser. SWAT typically runs on port 901, so it won't interfere with a web server if you want to run both on the same computer.

In order to use SWAT, you must first configure it to run. SWAT is normally launched from a super server (inetd or xinetd), so consult Chapter 9 for general super server information. If your distribution uses xinetd, look for a file called /etc/xinetd.d/swat and be sure it does not include a line that reads disable = yes. If your distribution uses inetd, check your /etc/inetd.conf file for a line like the following:

```
swat  stream  tcp  nowait  root  /usr/sbin/swat  swat
```

If this line, or one similar to it, is present but commented out by a hash mark (#), uncomment the line. If it's not present, add it. You might want to consider replacing the direct call to /usr/sbin/swat with a call to /usr/sbin/tcpd, the TCP Wrappers binary, which is described in Chapter 9. This will enable you to restrict access to SWAT by IP address; such restrictions are described in Chapter 7.

Whether you use inetd or xinetd, if you make changes to activate SWAT, you must reload or restart the super server, as described in Chapter 9. Typically, typing **/etc/inetd.d/xinetd reload** or something similar will do the trick.

Once SWAT is up and running, you can access it from a web browser running on the Samba server computer itself by entering a URL of `http://localhost:901`. The result is a prompt for a username and password. To configure Samba, you must type **root** and the `root` password. If you enter a normal username and password, you can view some information about the Samba server and change that user's Samba password, but you won't be able to do much else.

Once you're authenticated, SWAT presents a page with a series of documentation links and, near the top, a series of icons for performing various configuration actions: Home, Globals, Shares, Printers, Wizard, Status, View, and Password. The Home icon brings you back to the main page; the Globals, Shares, and Printers options let you configure global settings, file shares, and printer shares, respectively; the Wizard icon steps you through configuration in a friendly fashion; Status shows the status of the server; View presents the raw `smb.conf` file; and Password lets you change Samba passwords.

To configure global settings, file shares, or printer shares, click the appropriate link. The result is a page similar to the one shown in Figure 10.2, which shows the Shares page displaying information on a share. To select a share, pick it from the drop-down list next to the Choose Share button and then click that button, or create a new share by typing a name in the field next to the Create Share button and then clicking that button. You can then enter information on the share in the remaining fields. For still more options, click the Advanced button. When you've made all the changes you want to make, click Commit Changes. Nothing you enter will have any effect until you do so.

F I G U R E   1 0 . 2   SWAT enables you to configure Samba using a point-and-click interface.
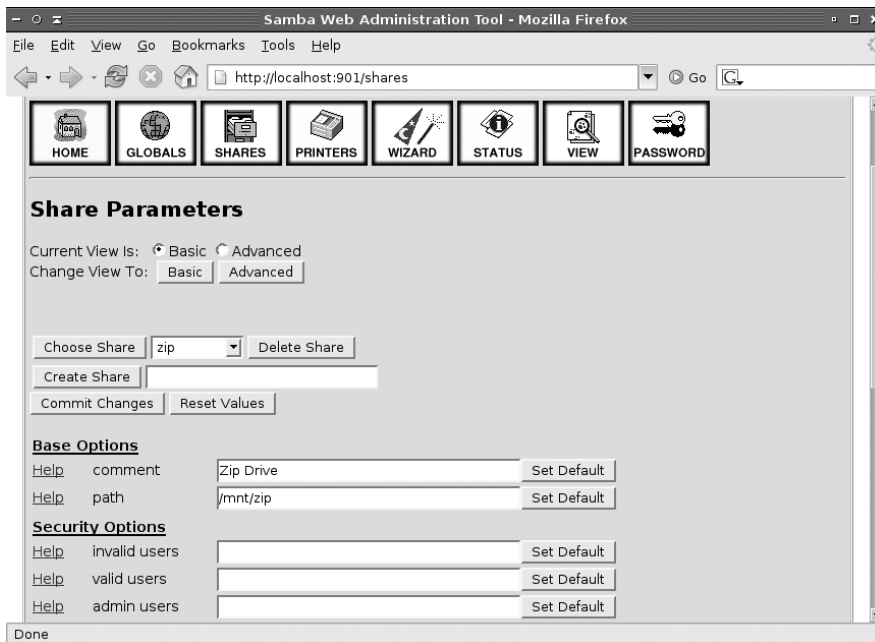
The SWAT options are named after the smb.conf parameters they control, so the earlier descriptions of smb.conf parameters should help you perform basic configuration with SWAT. If you're not sure what an option does, click the Help link next to the option. This action will open a new browser displaying the man page for smb.conf, scrolled to the relevant entry.

Keep in mind that SWAT, although a convenient way to administer Samba, is itself a server and therefore poses certain security risks. The password sent to SWAT is not encrypted, which makes it very risky to use it over the network, although using it from the Samba server computer itself is not so dangerous. You should definitely look into TCP Wrappers or xinetd options to restrict access to SWAT to the Samba server computer itself. You should only use SWAT for remote Samba administration on the most secure local networks.

🌐 Real World Scenario

Using WINS for Name Resolution

If your network is dominated by Windows computers, you might want to consider adding WINS to your regular Linux name resolution system. This will enable you to refer to computers by their NetBIOS names from Linux, even for TCP/IP applications such as e-mail clients. To do so, look for a line like the following in your /etc/nsswitch.conf file:

```
hosts:  files dns
```

Add wins to this line:

```
hosts:  files dns wins
```

Ordinarily, this is all you'll need to do, assuming you've set the correct workgroup name in your smb.conf file. This configuration does require that your system have the libnss_wins.so library installed (and a symbolic link called libnss_wins.so.2). Most distributions install this library, but a few don't or omit the symbolic link.

Once you've configured your system in this way, you should be able to access other systems by their NetBIOS names, even if they aren't listed by those names in a DNS server. This can be particularly handy if your network uses DHCP for IP address assignment and so there's no static mapping of IP addresses to hostnames.

Samba Security Considerations

Samba is a very powerful server and so can be a security liability if it's not administered properly. Some of the same caveats apply to Samba as apply to NFS. In particular, you should be wary about creating file shares to any sensitive directories, such as /etc. Such shares, if broken into, could be easily abused to give the intruder full access to the entire computer.

Samba's main form of access control is a username/password pair, unlike the IP address restrictions of NFS. This fact makes Samba somewhat less susceptible to intrusion merely because a

cracker has gained access to your local network wires. The usual configuration of using encrypted passwords also offers some protection against password cracking, even if an intruder manages to record the password exchange between two systems. Depending on the client and certain advanced Samba options, though, the encryption of SMB/CIFS encrypted passwords can be quite weak, so you shouldn't rely too much on the SMB/CIFS password encryption. As with NFS, non-password data is *not* encrypted, so you should be aware that potentially all the files stored on a Samba server (and files printed via Samba) could be intercepted. Overall, Samba is best used on at least moderately well protected local networks, not over the Internet at large or on a wireless network.

The Samba server itself is complex, and certain types of configuration can improve or degrade its security. Options related to passwords, IP-based access restrictions, and more are available. For more information, consult the main Samba documentation or a book on the server.

Using a Forwarding DNS Server

Chapter 9 described the basics of network configuration, including pointing a Linux system to a local DNS server. In some cases, though, you might want to run your own DNS server. A full-fledged DNS server is generally configured only to handle a domain you manage on the Internet, but smaller setups can be useful in other situations. Most notably, you might want to set up a caching (aka a forwarding) DNS server for a small network or even for a single system. Such a server receives DNS lookup requests, forwards them to another server, and caches the results for subsequent use. To enable such a server, you should first understand enough of the DNS protocol and of Linux DNS server software to know what you're managing. You can then set the options needed in the software to get a forwarding DNS server working. You must also be able to configure computers to use your caching name server. Finally, as with all servers, you should understand the security implications of running them.

DNS Server Basics

Most people don't give much thought to the DNS lookup process, but in fact, a great deal of complexity lurks just beneath the surface. This complexity involves both the structure of the DNS system and the design of the servers that manage it all.

Internet DNS Structure

In some sense, the complexity of DNS begins with the domain registration process. In order to avoid confusion, domain names can't simply be made up and used immediately. Imagine the chaos if several organizations tried to use a single domain name, such as `sybex.com` or `whitehouse.gov`. Perhaps different users would end up being directed to different sites or errors would result from conflicting lookups.

In order to impose order on the potential for chaos, domain names must be registered with a domain name registrar. Many registrars exist, but ultimately, each top-level domain (TLD), such as `.com` and `.gov`, has a single governing body that serves as a clearinghouse for all the domains within that TLD, such as `sybex.com` and `whitehouse.gov`.

Anybody with a few dollars can register an Internet domain. A list of registrars is available at `http://www.icann.org/registrars/accredited-list.html`. Go to one of the registrars on this list, enter the domain name you'd like to register on a form, and if the domain isn't already taken, pay the fee. Registration is typically about $15 per year (some registrars charge more, some less, and the fee often varies depending on the TLD you want). You may need to pay extra for DNS services for your domain if you don't provide your own DNS servers.

Name resolution involves a series of servers, each with detailed knowledge about different levels in the DNS hierarchy. At the top of this hierarchy are the *root servers,* which have IP addresses that change very infrequently. The root servers know the IP addresses of the DNS servers for the TLDs—`.com`, `.gov`, `.us`, and so on. When one computer (typically a DNS server) wants to know the IP address of another computer (say, `www.sybex.com`), the querying computer can ask one of the root servers to resolve that name. In most cases, the root server will reply with a message to the effect that it doesn't know the full name but that another computer (the `.com` DNS server, in this example) should know more. The querying computer then resubmits the original request to the `.com` DNS server. Chances are this server will also reply that it doesn't know the IP address, but that it has the address of a server with more information—the `sybex.com` DNS server, in this example. Upon querying this server, the original system will learn the IP address.

This process is known as a *recursive lookup,* and it can vary in its length. In theory, the root server could know the IP address of `www.sybex.com` and return it directly, or the lookup could involve more than the three DNS servers specified in this example, particularly for domains with more than three elements in the name (say, `challenger.taurus.littrow.apollo.luna.edu`).

The DNS server whose IP address you give to a computer as part of a static IP address configuration, as described in Chapter 9, typically performs a full recursive lookup. A shortcut, though, is to enable another DNS server to do the bulk of the work. If a DNS server is configured as a caching name server, it can pass the requests on to another server and ask that server to do the full lookup. The first DNS server then sits back, waits for the response, and passes it on to the client that asked for it. This approach can actually reduce network traffic and improve response times for a couple of reasons. First, small networks, such as those in small businesses, are often linked by relatively slow Internet connections. These networks' ISPs' network connections are usually much faster, so the ISPs' DNS servers can usually do the lookup much more quickly. Second, by caching popular results, the small networks' DNS servers can reply immediately rather than perform any lookup at all.

BIND Features

The most common name server in Linux is the *Berkeley Internet Name Domain (BIND),* which is also known as `named` (the name of the executable program). BIND is a very complex program, and entire books have been written about it. Consult a book such as Paul Albitz's and Cricket Liu's *DNS and BIND, 4th Edition* (O'Reilly, 2001) for full details.

BIND is controlled through several files. One entire set of files is related to domains that BIND is to manage. These files map hostnames to IP addresses and vice versa. This function of BIND, though, is beyond the scope of this book and of the LPIC exam, so you don't need to be very concerned with these files. Configuring a forwarding-only name server, though,

requires adjusting the primary BIND configuration file. For recent versions of BIND (version 8.*x* and 9.*x*), this file is called `named.conf`, and it's usually stored in `/etc`. For older versions of BIND (numbered 4.*x* and earlier; there were no versions between 4.*x* and 8.*x*), the main configuration file is called `named.boot`.

> As described in the upcoming section "BIND Security Considerations," many distributions now run BIND in a way that may affect the placement of its configuration file. If you don't see `named.conf` in `/etc` but you're sure the server is installed, use your package management tools to track it down.

Setting DNS Options for Your Network

Once you've installed BIND on your system, you can make a few small edits to configure the program as a forwarding-only server. With BIND 8.*x* or 9.*x*, load the `/etc/named.conf` file into your favorite editor and look for the `options` section, which probably resides at or near the start of the file. A forwarding configuration specifies one or more upstream DNS servers using the `forwarders` keyword:

```
options {
        directory "/var/named";
        forwarders {
                10.9.16.30;
                10.13.16.30;
                };
        forward only;
};
```

Chances are your configuration will lack the `forwarders` keyword, the two IP addresses surrounded by curly braces (`{}`), and the `forward only` line. Add these elements, but substitute the IP addresses of your upstream DNS servers, such as the IP addresses given to you by your ISP. (You can use just one server or several, but adding more servers can actually degrade performance in certain types of error conditions, so you should probably stick to two or three at most.) Pay careful attention to the punctuation in this example. With the exception of lines that end with an opening curly brace, all lines end in semicolons (`;`). The amount of white space is not critical, though, and in fact it's possible to list the IP addresses and closing curly brace on the same line as the `forwarders` keyword. (If the configuration is so compressed, you must still use semicolons after each IP address.)

The `forward only` line tells BIND to query the specified name servers and to return an error message if they fail to respond or return an error message themselves. An alternative is to specify `forward first`. Configured in this way, BIND attempts a lookup via the specified servers, but if they don't respond, BIND attempts a full recursive lookup. This configuration can be helpful in the event that a problem is isolated to the upstream DNS servers; however, it can impose further

delays in returning error codes to programs in the event of a more substantial problem, such as a complete failure of your upstream Internet connection.

You're unlikely to encounter a system that's so old that it runs the BIND 4.*x* server but the LPI objectives still refer to them. This old software uses the /etc/named.boot configuration file, and its syntax is different from that of /etc/named.conf. In particular, you can specify forwarding systems using a single punctuation-free line and forwarding options on another line:

```
forwarders 10.9.16.30 10.13.16.30
options forward-only
```

Of course, you should change the IP addresses to those for your upstream DNS server. You can omit the options line to implement the equivalent of a BIND 8/9 forward first configuration.

Once you've made these changes, you should reload or restart your BIND system. Because BIND is typically started via SysV startup scripts, you can do this by passing the reload or restart option to the appropriate script, as in **/etc/init.d/named reload**.

Using a Forwarding DNS Server

To do any good, clients must be able to access your forwarding DNS server. Chapter 9 describes basic Linux network configuration. The easiest case is static IP address assignment. If your network has no DHCP server, you can simply substitute the IP address of your forwarding server for the IP addresses of your upstream DNS servers in your local network systems' configurations. As described in Chapter 9, this is done by specifying nameserver lines in /etc/resolv.conf:

```
nameserver 192.168.1.1
```

If you've configured BIND as a caching-only name server for just one computer, you would point it to itself on this line.

If your network uses DHCP to assign IP addresses, you should reconfigure your DHCP server to deliver the IP address of your BIND server. If your DHCP server is the popular Linux Internet Software Consortium (ISC) DHCP server, you can edit the /etc/dhcpd.conf file. Look for a line like the following:

```
option domain-name-servers 10.9.16.30;
```

Change the IP address (or IP addresses, if more than one is listed) to point to your local network's new forwarding DNS server.

However the DNS server is set, you should also be aware of the importance of the /etc/nsswitch.conf file. This file controls the Name Service Switch (NSS), which manages certain types of system lookups, including hostname/IP address translations. As described in the sidebar "Using WINS for Name Resolution" earlier in this chapter, it's possible to reconfigure what tools are used for performing name lookups. Most configurations include a hosts line that specifies files and dns for name lookups. The dns entry is the critical one for performing DNS lookups, while files refers to the /etc/hosts file, which is described in Chapter 9.

NSS is designed to be expandable. It's possible that some future system will replace `files` or `dns` on the `hosts` line but still use DNS for hostname resolution. If you find something you don't recognize on this line but DNS resolution seems to work, I recommend you don't try to adjust the entry until you learn what the entries you see are supposed to do.

BIND Security Considerations

BIND provides relatively limited access to the computer—it reads just a few files and doesn't give most users any ability to write to the local filesystem. (Exceptions exist for some advanced features, such as automated domain updates.) These factors tend to limit the possibility for abuse via BIND. On the other hand, like Apache, BIND requires no authentication, and BIND is a very complex server. These factors tend to increase the risk associated with BIND.

For a forwarding-only name server, the single most important security concern is to isolate the server from the Internet. Use firewall rules (on the BIND server itself or on your router) to prevent outside systems from initiating contact with the BIND ports (TCP and UDP port 53). This procedure should help prevent the server from being abused by miscreants on the Internet at large.

To reduce the risks associated with BIND, many distributions now ship the server to run from something known as a *chroot jail*. This is a way of running a server so that it doesn't have full access to the computer's filesystem, even as `root`. The server "thinks" that some directory (say, `/var/lib/named`) is actually the root directory (`/`). Configuring a chroot jail is beyond the scope of this book; I mention it mainly because you may need to look for your BIND configuration files in a peculiar location, such as `/var/lib/named/etc`, if your system is configured in this way. Some distributions set up links to make this process easier, but this isn't guaranteed.

Configuring SSH

In the past, Telnet has been the remote text-mode login protocol of choice on Linux and Unix systems. Unfortunately, Telnet is severely lacking in security features. Thus, in recent years SSH has grown in popularity, and is in fact the preferred remote login tool. SSH can also handle file transfer tasks similar to those of the File Transfer Protocol (FTP). For these reasons, knowing how to configure SSH can be very helpful. This task requires knowing a bit about SSH generally and about the SSH configuration file under Linux. As is usual in this chapter, I conclude the look at SSH with information on the security implications of running the server.

SSH is not as complex as some servers described in this chapter, but it's still complex enough that I can't cover more than its basics in this chapter. Consult OpenSSH's own documentation or a book on the topic, such as *SSH, The Secure Shell: The Definitive Guide, Second Edition*, by Daniel J. Barrett, Richard Silverman, and Robert G. Byrnes (O'Reilly, 2005), for more details.

SSH Basics

Linux supports remote login access through several different servers, including Telnet, Virtual Network Computing (VNC), and even X. Unfortunately, most of these methods suffer from a major drawback: They transfer all data over the network in unencrypted form. This fact means that anybody who can monitor network traffic can easily snatch sensitive data, often including passwords. (VNC and a few other protocols encrypt passwords but not other data, though.) This limitation puts a serious dent in the utility of these remote login tools; after all, if using a remote access protocol means you'll be giving away sensitive data or even compromising your entire computer, it's not a very useful protocol.

> **WARNING** Non-encrypting remote access tools are particularly risky for performing work as root, either by logging in directly as root or by logging in as an ordinary user and then using su, sudo, or other tools to acquire root privileges.

SSH was designed to close this potentially major security hole by employing strong encryption techniques for all parts of the network connection. SSH encrypts the password exchange and all subsequent data transfers, making it a much safer protocol for remote access.

In addition to encryption, SSH provides file transfer features and the ability to *tunnel* other network protocols—that is, to enable non-encrypted protocols to "piggyback" their data over an SSH connection, thus delivering SSH's encryption advantages to other protocols. This feature is frequently employed in conjunction with X, enabling encrypted remote GUI access, as described in Chapter 5, "The X Window System."

Of course, SSH's advantages don't come without a price. The main drawback of SSH is that the encryption and decryption consume CPU time. This fact slows down SSH connections compared to those of direct connections and can degrade overall system performance. This effect is modest, though, particularly for plain text-mode connections. If you tunnel a protocol that transfers much more data, such as X, you may see a greater performance drop when using SSH. Even in this case, the improved security is generally worth the slight performance drop.

Several SSH servers are available for Linux, but the most popular by far is the OpenSSH server (`http://www.openssh.org`). This program was one of the first open-source implementations of the SSH protocol, which was developed by the commercial SSH Communications (`http://www.ssh.com`), whose server is now sold under the name SSH Tectia. OpenSSH, SSH Tectia, and other SSH products can interoperate with one another, assuming they're all configured to support at least one common level of the SSH protocol. OpenSSH 4.0, the latest version as I write, supports SSH levels 1.3, 1.5, and 2.0, with 2.0 being the preferred level because of known vulnerabilities in the earlier versions.

> **NOTE** OpenSSH is closely associated with the OpenBSD OS, so its website has an Open-BSD bias. If you visit the site, you may want to click the Linux link under the Other OSs heading. You can find Linux-compatible source code and binaries from that site, and OpenSSH now ships with most Linux distributions.

OpenSSH may be launched via either a super server (`inetd` or `xinetd`) or a SysV startup script. The latter method is preferred, though, because the server may need to perform some CPU-intensive tasks upon starting, so if it's started from a super server, OpenSSH may be sluggish to respond to connection requests, particularly on systems with weaker CPUs. Most distributions deliver SysV startup scripts with their SSH packages. If you make changes to your SSH configuration, you may need to pass the `reload` or `restart` option to the startup script, as in **/etc/init.d/sshd reload**. (Chapter 6 covers SysV startup scripts in more detail.) However it's launched, the OpenSSH server binary name is `sshd`—the same as the binary name for SSH Tectia.

Setting SSH Options for Your System

For the most part, SSH works reasonably well when it's first installed, so you may not need to make any changes to its configuration. If you do need to make changes, though, these are mostly handled through the main SSH configuration file, /etc/ssh/sshd_config. You can also edit some additional files to limit access to the SSH server or to change how SSH manages the login process.

Configuring Basic SSH Features

The /etc/ssh/sshd_config file consists mainly of option lines that take the following form:

Option value

WARNING   Don't confuse the sshd_config file with the ssh_config file. The former controls the OpenSSH server, whereas the latter controls the SSH client program, ssh.

In addition to configuration lines, the `sshd_config` file holds comments, which are denoted by hash marks (#). Most sample configuration files include a large number of SSH options that are commented out; these lines specify the default values, so uncommenting the lines without otherwise changing them will have no effect. If you want to change an option, uncomment the line and change it. Most options' default values are suitable for most systems. The following list includes some that you might want to check and, perhaps, change:

Protocol   This option specifies the protocol levels OpenSSH understands. Possible values are 1 and 2. You can configure OpenSSH to support both protocols by separating them by a comma, as in 1,2 or 2,1, which are equivalent. Given the fact that OpenSSH protocol level 1 has been compromised, the safest configuration is to set Protocol 2. This will limit the server's ability to communicate with older clients, though.

PermitRootLogin   By default, this option is set to yes, which enables OpenSSH to accept direct logins by root. This is safer than a similar configuration under Telnet, but for a bit of added security, set this value to no. The result will be that anybody wanting to perform remote work as root will need to first log in as an ordinary user, which means that an intruder who has somehow acquired the root password will also need a regular username and its password.

X11Forwarding   This option specifies whether or not OpenSSH's X tunneling features should be active. If you want to enable remote users to run X programs via SSH, you must set this option to yes. Doing so can slightly degrade security of the client's X display, though, depending on certain other options, hence the conservative default value of no.

For information on additional options, consult the man page for sshd_config. If you make changes to the SSH configuration, remember to restart it using the server's SysV startup script.

SSH Keys

Part of SSH's security involves *encryption keys*. Each server system and each user has a unique number, or key, for identification purposes. In fact, SSH uses a security system that involves two keys: a *public key* and a *private key*. These two keys are mathematically linked in such a way that data encrypted with a particular public key may be decrypted only with the matching private key. When establishing a connection, each side sends its public key to the other. Thereafter, each side encrypts data with the other side's public key, ensuring that the data can be decrypted only by the intended recipient. In practice, this is just the first step of the process, but it's a very critical one. What's more, SSH clients typically retain the public keys of servers they've contacted. This enables them to spot changes to the public key. Such changes can be signs of tampering.

Most OpenSSH server SysV startup scripts include code that looks for stored public and private keys and, if they're not present, generate them. In total, six keys are needed: public and private keys for three encryption tools SSH supports. These keys are normally stored in /etc/ssh and take filenames of the form /etc/ssh/ssh_host*key and /etc/ssh/ssh_host*key.pub for private and public keys, respectively. If your system doesn't have these keys and you can't seem to get the SSH server to start up, you can try generating the keys with the ssh-keygen command:

```
# ssh-keygen -q -t rsa1 -f /etc/ssh/ssh_host_key -C '' -N ''
# ssh-keygen -q -t rsa -f /etc/ssh/ssh_host_rsa_key -C '' -N ''
# ssh-keygen -q -t dsa -f /etc/ssh/ssh_host_dsa_key -C '' -N ''
```

Each of these commands generates both a private key (named in the -f parameter) and a public key (with the same name but with .pub appended).

Do not run these ssh-keygen commands if the SSH key files already exist. Replacing the working files will cause clients who've already connected to the SSH server to complain about the changed keys and possibly refuse to establish a connection.

WARNING   Be sure that the non-public keys are suitably protected; if an intruder obtains one of these keys, the intruder can impersonate your system. Typically, these files should have 0600 (-rw-------) permissions and be owned by root. The public key files (with .pub filename extensions) should be readable to all users, though.

When you configure a client system, you might want to consider creating a global cache of host keys. As already noted, the `ssh` program records host keys for each individual user. (It stores these in the `~/.ssh/known_hosts` file.) When you set up the client, though, you can populate the global `ssh_known_hosts` file, which is normally stored in `/etc` or `/etc/ssh`. Doing so will ensure that the public key list is as accurate as the sources you use to populate the global file. It will also eliminate confirmation messages when users connect to the hosts whose keys you've selected to include in the global file.

How do you create this file, though? One simple way is to copy the file from a user account that's been used to connect to the servers you want to include. For instance, you might type **cp /home/ecernan/.ssh/known_hosts /etc/ssh/ssh_known_hosts** to use `ecernan`'s file. You may want to manually review this file before copying it, though. It consists of one line per host. Each line begins with a hostname, IP address, or both, and continues with the key type and the key. You can ignore most of this information, but pay attention to the hostnames and IP addresses. Ensure that the list includes all the SSH servers your client is likely to want to use and that it does *not* include inappropriate or unnecessary servers. You can remove lines in your text editor, if necessary. To add entries, use the account whose file you're copying to connect to the system you want to add. Chances are `ssh` will display a warning about connecting to an unknown system. Confirm that you want to do so. Once you do this and reload the file, you should see an entry for the server.

Controlling SSH Access

You can limit who may access an SSH server in various ways. The most obvious and basic method is via password authentication. The usual SSH authentication method is to employ a username and password, much as Telnet does. (The `ssh` client program sends the username automatically or as part of the command line, though, so you won't see a username prompt when logging in via `ssh`.)

Beyond password authentication, SSH supports several other types of limitations:

TCP Wrappers   If you run SSH from a super server or if the server was compiled with TCP Wrappers support, you can use the `/etc/hosts.allow` and `/etc/hosts.deny` files to limit access by IP address. Chapter 7 describes TCP Wrappers in more detail. Note that if you launch SSH via a SysV startup script, this approach only works if the server was compiled to support it. This support might or might not be present in your distribution's standard SSH package.

Firewalls   As with all servers, you can restrict access by using a firewall, as described in Chapter 7. SSH uses TCP port 22. Technically, this isn't an SSH feature, but it's certainly useful for protecting an SSH server.

/etc/nologin   If this file is present, SSH honors it. As described in Chapter 7, this file's presence means that only `root` may log in. Ordinarily, the file's contents are displayed as an error message; however, OpenSSH doesn't do this.

SSH Login Scripts

Ordinarily, an SSH text-mode login session runs the user's configured shell, which runs the shell's defined login scripts, as described in Chapter 6. The OpenSSH server also supports its

own login script, sshrc (normally stored in /etc or /etc/ssh). The OpenSSH server runs this script using /bin/sh, which is normally a symbolic link to bash, so you can treat it as an ordinary bash script.

SSH Security Considerations

SSH is intended to solve security problems rather than create them. Indeed, on the whole using SSH is superior to using Telnet for remote logins, and SSH can also take over FTP-like functions and tunnel other protocols. Thus, SSH is a big security plus compared to using less-secure tools.

Like all servers, though, SSH can be a security liability if it's run unnecessarily or inappropriately. Ideally, you should configure SSH to accept only protocol level 2 connections and to refuse direct root logins. If X forwarding is unnecessary, you should disable this feature. If possible, use TCP Wrappers or a firewall to limit the machines that can contact an SSH server. As with all servers, you should keep SSH up-to-date; there's always the possibility of a bug causing problems.

You should consider whether or not you really need a remote text-mode login server. Such a server can be a great convenience—often enough to justify the modest risk involved. For extremely high-security systems, though, using the computer exclusively from the console may be an appropriate approach to security.

One unusual security issue with SSH is its keys. As noted earlier, the private key files are extremely sensitive and should be protected from prying eyes. Remember to protect the backups of these files, as well—don't leave a system backup tape lying around where it can be easily stolen.

Summary

Linux can be a powerful server OS. Common server programs for Linux include the sendmail mail server, Apache web server, NFS and Samba file servers, BIND (aka named) DNS server, and SSH remote login server. Each of these servers has its own unique configuration files, options, and security risks. Most share certain commonalities, though, such as the fact that bugs or misconfiguration can lead to compromised systems. You should always be cautious when configuring a server and keep the server software up-to-date lest an intruder take advantage of a problem to gain control of the server or, worse, of the entire computer.

Exam Essentials

Summarize the procedure for configuring sendmail.   Sendmail must be configured through its /etc/mail/sendmail.cf file, but this file's format is very tedious. Therefore, most administrators create another type of file that has a more manageable structure and create a sendmail.cf file using the m4 utility.

Explain why an open mail relay configuration is undesirable.   Open mail relays are computers that are configured to accept e-mail from any system and deliver it to any other system. These systems can be abused by spammers to help hide their identities.

Describe common locations of the Apache configuration files.   Apache configuration files are usually called `http.conf`, `httpd2.conf`, `apache.conf`, or `apache2.conf`. These files are usually located in `/etc/apache`, `/etc/apache2`, `/etc/httpd`, `/etc/httpd2`, or the `conf` subdirectory of any of these. Unfortunately, cross-distribution consistency on this matter remains elusive.

Determine where Apache looks for files it delivers to clients.   The `DocumentRoot` directive tells Apache where to look for files on the main site; this directive specifies the complete path to the root of the directory tree used for the files it delivers. You can also tell Apache where to look within users' home directories for user web pages with the `UserDir` directive.

Summarize the format of the `/etc/exports` file.   The `/etc/exports` file controls the directories that are shared via NFS. It consists of one line per directory. Each line begins with a directory path and includes the name or IP address of every client or group of clients that should have access to the export, with options enclosed in parentheses after the IP address or hostname.

Describe the difference between configuring Samba file and printer shares.   Samba file and printer shares are virtually identical from a configuration point of view; the main difference is the `printable = Yes` or `print ok = Yes` parameter, which identifies a printer share. Although printer shares are in some sense read/write shares, they need not be explicitly configured as such.

Explain where Samba and NFS are best deployed.   Samba is an implementation of the SMB/CIFS protocol suite, which is most often used for file and printer sharing by Windows systems; thus, Samba is best used as a server for Windows clients. NFS, by contrast, was designed as a file-sharing protocol for Unix systems, so it's best used for sharing files with Unix or Linux clients.

Describe the purpose of root DNS servers.   Root DNS servers deliver information on the IP addresses of servers responsible for handling top-level domains (TLDs)—`.com`, `.net`, `.uk`, and so on. Finding this information is the first step in your local DNS server's job of converting a hostname into an IP address.

Summarize the advantages of running a forwarding DNS server.   A forwarding DNS server passes whole DNS requests from a local network (or even just one computer) to an upstream DNS server. The main advantage is that the forwarding DNS server can then cache the results, improving lookup speed on subsequent requests.

Explain why SSH is preferred to Telnet for remote text-mode logins.   The Secure Shell (SSH) protocol provides encryption for all traffic, including both the password exchange and all subsequent data exchanges, whereas Telnet does not. This makes SSH much safer (if not 100 percent safe) for the exchange of sensitive data, particularly over untrusted networks such as the Internet.

Identify the most important SSH configuration file.   The SSH server is controlled through the `/etc/ssh/sshd_config` file. The SSH client configuration file is `/etc/ssh/ssh_config`; don't confuse the two.

Review Questions

1. Which of the following commands would you type to see if the mail service is functioning and view a backlog of old messages?

   A. `postfix`

   B. `traceroute`

   C. `sendmail`

   D. `mailq`

2. What will be the effect of the following lines in a sendmail `m4` configuration file?

   ```
   MASQUERADE_AS(`example.org')
   FEATURE(masquerade_envelope)
   ```

   A. Mail addressed to users in the `example.org` domain will be hidden from view until the postmaster can examine it.

   B. The server will change all domain name references in all e-mail headers to `example.org`, for both incoming and outgoing e-mail.

   C. Mail sent through the server will be identified as coming from `example.org`, but only if the mail's return address omits a domain name.

   D. Mail sent through the server will be identified as coming from `example.org`, even if the user has set another address in a mail client.

3. Which file might you edit to have mail addressed to `gertrude` redirected to `gerty`? (Select all that apply.)

   A. `/var/spool/mail/gertrude`

   B. `newnames` in `/etc` or `/etc/mail`

   C. `aliases` in `/etc` or `/etc/mail`

   D. `.forward` in `gertrude`'s home directory

4. Your network's Internet connection went down an hour ago and has only recently come back up. How would you check to see how many outgoing mail messages have been stuck on your local sendmail server because of this problem?

   A. Type `sendmail -q`.

   B. Type `mailq`.

   C. Type `xmqueue` and click the Queued Messages button.

   D. Log in as `postmaster` and type `pqcheck`.

5. What is the name of the protocol Apache uses to communicate with clients?

   A. The Patchy Server Protocol

   B. The Hypertext Transfer Protocol

   C. The File Transfer Protocol

   D. The Hypertext Markup Language

6. In what directory will you find the files delivered by an Apache server to its clients?

   A. /var/www/htdocs

   B. /var/www/localhost/htdocs

   C. /var/apache/htmlfiles

   D. The location is not consistent across systems.

7. What port should you block with a firewall if you want to limit access to a web server running on its standard port?

   A. TCP port 22

   B. TCP port 25

   C. TCP port 80

   D. TCP port 901

8. How does an NFS server determine who may access files it's exporting?

   A. It uses the local file ownership and permission in conjunction with the client's user authentication and a list of trusted client computers.

   B. It uses a password that's sent in unencrypted form across the network.

   C. It uses a password that's sent in encrypted form across the network.

   D. It uses the contents of individual users' .rlogin files to determine which client computers may access a share.

9. What is the effect of the following line in /etc/fstab?

   ```
   share:/server  /mnt/samba  nfs  defaults  0 0
   ```

   A. It mounts the SMB/CIFS share called server from the server called share at /mnt/samba.

   B. It mounts the SMB/CIFS share called share from the server called server at /mnt/samba.

   C. It mounts the NFS export called share from the server called server at /mnt/samba.

   D. It mounts the NFS export called server from the server called share at /mnt/samba.

10. Which of the following parameters should you definitely check and, if necessary, adjust in the [global] section of your smb.conf file when configuring Samba?

   A. path

   B. workgroup

   C. comment

   D. write list

11. Joe is editing an smb.conf file and spots the line print ok = Yes in a share definition. He changes this line to read printable = Yes. What effect will this change have on the operation of the share?

   A. It converts a printer share into a file share.

   B. It converts a file share into a printer share.

   C. It makes Samba report errors correctly when they occur.

   D. None of the above.

12. Why might you want to run SWAT?

    A. It provides name resolution using NetBIOS protocols.

    B. It provides Web-based GUI administration of a Samba server.

    C. It's run from the `inetd` or `xinetd` super server, unlike many servers.

    D. It helps to find and correct bugs in a wide variety of servers.

13. How many individual queries are involved in a full recursive DNS lookup?

    A. 1

    B. 3

    C. 7

    D. A variable number

14. You compare two `/etc/named.conf` files and find that they differ in only one line. The first includes a line in the `options` section that reads `forward first;`, while the second contains a line that reads `forward only;`. Assuming the rest of the forwarding name server configurations in these files are correct, what is the effect of this small difference?

    A. The `forward first;` configuration performs a forwarding lookup and then does a full recursive lookup if the forwarding lookup fails; the `forward only;` configuration performs a forwarding lookup but never attempts a full recursive lookup.

    B. The `forward first;` configuration forwards DNS information to clients before logging the data; the `forward only;` configuration forwards DNS information to clients without logging information on the lookup.

    C. The `forward first;` configuration does a forwarding lookup and checks the results against a full recursive lookup; the `forward only;` configuration performs a forwarding lookup but never attempts a full recursive lookup.

    D. None of the above; neither configuration is valid.

15. In what file would you enter the following line to use a forwarding DNS server you've set up on 192.168.1.1?

```
nameserver 192.168.1.1
```

    A. `/etc/named.conf`

    B. `/etc/nameservers.conf`

    C. `/etc/resolv.conf`

    D. `/etc/dnsrc`

16. Under what circumstances should a forwarding-only name server be accessible to the Internet at large?

    A. Rarely or never

    B. Almost always

    C. Only when your domain already has two other DNS servers on the Internet

    D. Only when you use BIND 8.*x* or later

17. Which servers might you consider retiring after activating an SSH server? (Select all that apply.)

    A. SMTP

    B. Telnet

    C. FTP

    D. NFS

18. You find that the `ssh_host_dsa_key` file in `/etc/ssh` has 0666 (`-rw-rw-rw`) permissions. Your SSH server has been in operation for several months. Should you be concerned?

    A. Yes

    B. No

    C. Only if the `ssh_host_dsa_key.pub` file is also world-readable

    D. Only if you're launching SSH from a super server

19. For best SSH server security, how should you set the `Protocol` option in `/etc/ssh/sshd_config`?

    A. `Protocol 1`

    B. `Protocol 2`

    C. `Protocol 1,2`

    D. `Protocol 2,1`

20. Why is it unwise to allow `root` to log on directly using SSH?

    A. Somebody with the `root` password but no other password could then break into the computer.

    B. The `root` password should never be sent over a network connection; allowing `root` logins in this way is inviting disaster.

    C. SSH stores all login information, including passwords, in a publicly readable file.

    D. When logged on using SSH, `root`'s commands can be easily intercepted and duplicated by undesirable elements.

Answers to Review Questions

1. D. The `mailq` utility can display a backlog of old messages and show you if the mail service is functioning. Postfix and sendmail are both mail server programs, and their executable commands won't perform this function. The `traceroute` program is an enhanced version of `ping` that shows the route data takes to reach a target, as described in Chapter 9.

2. D. Option D correctly describes the effect of these lines. Option A describes the effect of some advanced spam- and virus-fighting tools, but not of the specified lines. Option B goes too far; hostname masquerading applies only to the `From:` address line on outgoing mail, not to all headers. Option C correctly describes the effect of the first line alone but not of both lines together.

3. C, D. The `aliases` file in `/etc` or `/etc/mail` sets up e-mail aliases, which are effectively alternative e-mail names for accounts. A user's `~/.forward` file holds information that causes messages to the account to be forwarded to another account or address. Both approaches can accomplish the stated goal, although the `~/.forward` file requires that the `gertrude` account exists and has a home directory. The `/var/spool/mail/gertrude` file is one possible location for mail addressed to `gertrude` to be stored by the mail server, but editing this file won't do any good. The `newnames` file of option B is fictitious.

4. B. The `mailq` program returns information on the mail queue, including how many messages are awaiting delivery. You might type **`sendmail -q`**, as in option A, to clear the queue quickly, but this command won't actually tell you how many messages are waiting in the queue. The `xmqueue` and `pqcheck` commands of options C and D are both fictitious.

5. B. The Hypertext Transfer Protocol (HTTP) is the name of the protocol Apache uses and is the source of the `http://` code in front of web page names. Option A is fictitious, although it's related to a pun that gave rise to the server name *Apache* (because the server had been greatly patched). Option C, the File Transfer Protocol (FTP), is the name of a common protocol for transferring files, but it's not one that's implemented by Apache. Option D, the Hypertext Markup Language (HTML), is a common format for files delivered by Apache, but this isn't the protocol used by the server.

6. D. No one directory is a correct answer to this question because the directory is set with the `DocumentRoot` directive in the Apache configuration file.

7. C. Web servers run on TCP port 80 by default. TCP port 22 is the Secure Shell (SSH) port, TCP port 25 is the Simple Mail Transfer Protocol (SMTP) port, and TCP port 901 is used by the Samba Web Administration Tool (SWAT).

8. A. NFS uses a "trusted host" policy to let clients police their own users, including access to the NFS server's files. NFS does not use a password, nor does it use the `.rlogin` file in users' home directories.

9. D. Option D correctly describes the effect of this `/etc/fstab` entry. The server, share, and mount point names are all deceptive in this example. Options A and B both incorrectly state that the share is an SMB/CIFS share, when in fact it's an NFS export, as indicated by both the format of the server identification and by the use of an `nfs` filesystem type code. Option C is closer to correct, but it confuses the role of the server and export; in an NFS specification, the server name comes first, followed by a `:/` symbol pair and the name of the export.

10. B. The `workgroup` parameter identifies the name of the workgroup or domain to which the Samba server belongs. This is a global option that must be set correctly for Samba to interact normally with other SMB/CIFS clients and servers. The `path`, `comment`, and `write list` parameters are all share-level parameters that describe individual shares. Although you could theoretically find any of them in the `[global]` section as a way of setting defaults for subsequent shares, they aren't as necessary for basic Samba functioning, and they can all be overridden within individual share definitions.

11. D. The `print ok` and `printable` parameters are synonyms, so Joe's change has no effect on the operation of the share. Both before and after the change, the share was a printer share. Errors are handled in precisely the same way, no matter which parameter name is used.

12. B. SWAT is the Samba Web Administration Tool, which provides local or remote GUI administration for a Samba server. Option A describes certain configuration changes to `/etc/nsswitch.conf`, but SWAT is unnecessary for making these changes. Option C is a true statement, but the fact that SWAT is run from a super server is not sufficient reason to run it. Option D is simply a made-up description; it doesn't describe SWAT.

13. D. Recursive DNS lookups take an unknown number of queries to various DNS servers that handle different parts of the DNS name space.

14. A. Option A correctly describes the difference between these two configuration options. Option B is a completely fictitious description. Option C is almost correct, but `forward first;` does not result in a full recursive lookup unless the forwarding lookup fails in certain ways. Both configurations are valid, at least within the context of an otherwise valid BIND 8/9 configuration.

15. C. The `/etc/resolv.conf` file holds the IP addresses of the DNS servers your system should use, in the format of the specified line; thus, option C is correct. Option A, `/etc/named.conf`, is the main BIND configuration file on the DNS server itself. Options B and D are both fictitious files.

16. A. A forwarding-only name server is normally intended for use only by the systems on a local network. Exposing such a system to the Internet at large is almost certainly an unnecessary security risk, so option A is correct. Option B is the exact opposite of the correct answer. Option C is incorrect because the number of existing DNS servers you have serving your domain is irrelevant to the question of forwarding DNS servers. The version of BIND, as mentioned in option D, is also unimportant to the question, at least as a matter of principle. (As with all servers, some versions have known security bugs, of course.)

17. B, C. SSH is most directly a replacement for Telnet, but SSH also includes file transfer features that enable it to replace FTP in many situations. SSH is not a direct replacement for either SMTP or NFS.

18. A. The `ssh_host_dsa_key` file holds one of three critical private keys for SSH. The fact that this key is readable (and writeable!) to the entire world is disturbing. In principle, a miscreant who has acquired this file might be able to redirect traffic and masquerade as your system, duping users into delivering passwords and other sensitive data.

19. B. SSH protocol level 2 is more secure than protocol level 1; thus, option B (specifying acceptance of level 2 only) is the safest approach. Option A is the *least* safe approach because it precludes the use of the safer level 2. Options C and D are exactly equivalent in practice; both support both protocol levels.

20. A. Allowing only normal users to log in via SSH effectively requires two passwords for any remote `root` maintenance, improving security. SSH encrypts all connections, so it's unlikely that the password, or commands issued during an SSH session, will be intercepted. (Nonetheless, some administrators prefer not to take even this small risk.) SSH doesn't store passwords in a file.

Glossary

Numbers

8.3 filename   A filename that consists of no more than eight characters plus an optional dot (.) and three-character extension. This file naming limit exists in DOS and the original File Allocation Table (FAT) filesystem it uses.

A

access control list (ACL)   A security system that provides a list of usernames or groups and their permissions to access a resource. ACLs are expanding and supplementing traditional Unix-style permissions on new filesystems. Ext2fs, ext3fs, JFS, and XFS all support ACLs natively, and ACL extensions for ReiserFS are available.

account   Stored information and a reserved directory that allows one individual to use a computer. The term is often used and thought of as if it were a distinct virtual component of a computer that a person can use, as in "Sam logged into his account," or "Miranda's account isn't working."

ACL   See *access control list (ACL)*.

active partition   The partition that's marked as bootable in the Master Boot Record (MBR). Some boot loaders, such as the standard DOS/Windows boot loader, boot the active partition.

Address Resolution Protocol (ARP)   A protocol used to learn a network hardware address based on an IP address.

Advanced Linux Sound Architecture (ALSA)   One of two major sets of sound drivers in Linux. Added as a standard part of the 2.6.*x* kernel series, but available as an add-on package for earlier kernels. See also *Open Sound System (OSS)*.

Advanced Technology Attachment (ATA)   A type of interface for hard disks, CD-ROM drives, tape drives, and other mass storage devices. Also often referred to as *EIDE*.

ALSA   See *Advanced Linux Sound Architecture (ALSA)*.

anti-aliasing   A technique for rendering fonts that uses shades of gray along curved or angled edges in order to improve the legibility and aesthetic appeal of individual characters.

Apache   A common web server package for Linux.

ARP   See *Address Resolution Protocol (ARP)*.

ATA   See *Advanced Technology Attachment (ATA)*.

B

backport   The practice of moving more advanced features into less advanced versions of a package. This has been common to add hardware support from development kernels into stable kernels.

Basic Input/Output System (BIOS)   A low-level software component included on a computer's motherboard in read-only memory (ROM) form. The CPU runs BIOS code when it first starts up, and the BIOS is responsible for locating and booting an OS or OS loader.

baud   A measure of data transmission speed, commonly used over serial lines, corresponding to the number of signal elements transmitted per second. This term is often used as a synonym for "bits per second," but many modems encode more than one bit per signal element, so the two aren't always synonymous.

Berkeley Internet Name Domain (BIND)   A common Domain Name System (DNS) server for Linux.

BIND   See *Berkeley Internet Name Domain (BIND)*.

BIOS   See *Basic Input/Output System (BIOS)*.

bitmap font   A font whose characters are defined in terms of the activation or inactivation of individual pixels. Bitmap fonts are quick to display but inflexible. See also *scalable font*.

boot loader   A program that directs the boot process. The BIOS calls the boot loader, which loads the Linux kernel or redirects the boot process to another boot loader. Also known as a *boot manager*.

boot manager   See *boot loader*.

broadband   1. High-speed (greater than 200Kbps) Internet connections delivered to homes and small businesses. 2. Networking technologies that support simultaneous transmission of data, voice, and video.

broadcast   A type of network access in which one computer sends a message to many computers (typically all the computers on the sender's local network segment).

build number   A number identifying minor changes made to a binary package by its maintainer rather than changes implemented by the program's author, which are reflected in the version number.

C

C library (libc)   Standard programming routines used by many programs written in the C programming language. The most common Linux C library is also referred to as *GNU libc (glibc)*.

cathode ray tube (CRT)   A type of computer display that uses a glass screen with an electron gun that shoots charged particles at the screen to make images. CRTs are similar to conventional television sets, but they're declining in popularity in favor of LCD monitors.

central processing unit (CPU)   The main chip on a computer; it handles the bulk of its computational tasks.

CGI   See *Common Gateway Interface (CGI)*.

checksum   A simple file integrity check in which the values of individual bits or bytes are summed up and compared to a stored value for a reference version of the file.

chroot jail   A method of running a program (particularly a server) so that it's access to the computer is limited to a particular directory tree. A chroot jail is a useful security measure for certain types of servers.

CHS geometry   See *cylinder/head/sector (CHS) geometry*.

CIDR   See *classless inter-domain routing (CIDR)*.

classless inter-domain routing (CIDR)   A method of breaking down IP addresses into subnets for routing purposes that does not rely on the traditional Class A/B/C distinctions. CIDR is more flexible than the class system but requires certain Internet routers to have larger routing tables.

client   1. A program that initiates data transfer requests using networking protocols. 2. A computer that runs one or more client programs.

CMOS   See *complementary metal oxide semiconductor (CMOS)*.

command completion   A feature of many Linux shells that simplifies typing long commands. Pressing the Tab key causes the shell to search for possible commands or filenames that would complete the command. If only one command or filename matches the characters typed so far, the shell completes the entry. If not, the shell enters the characters up to the point where the user must specify another character.

Common Gateway Interface (CGI)   A system for running scripts or programs from a web server at the request of web clients.

Common Unix Printing System (CUPS)   A relatively recent printing system for Linux and other Unix-like systems. CUPS adds several features that had been missing from the earlier BSD LPD and LPRng printing systems.

complementary metal oxide semiconductor (CMOS)   A part of the BIOS that gives the user the ability to control key chipset features, such as enabling or disabling built-in ports.

conditional expression   A construct of computer programming and scripting languages used to express a condition, such as the equality of two variables or the presence of a file on a disk. Conditional expressions enable a program or script to take one action in one case and another action in the other case.

Coordinated Universal Time (UTC)   A time closely related to *Greenwich Mean Time (GMT)*.

CPU   See *central processing unit (CPU)*.

cracker   An individual who breaks into computers. Crackers may do this out of curiosity, malice, for profit, or for other reasons.

cron job   A program or script that's run at a regular interval by the `cron` daemon. See also *system cron job* and *user cron job*.

CRT   See *cathode ray tube (CRT)*.

CUPS   See *Common Unix Printing System (CUPS)*.

cylinder/head/sector (CHS) geometry   A method of hard disk addressing in which a triplet of numbers (a cylinder, a head, and a sector) are used to identify a specific sector. CHS geometry addressing contrasts with linear block addressing (LBA).

D

daemon   A program that runs constantly, providing background services. Linux servers are typically implemented as daemons, although there are a few nonserver daemons.

Data Display Channel (DDC)   A protocol that enables a computer to query a monitor for its maximum horizontal and vertical refresh rates and other vital statistics.

DDC   See *Data Display Channel (DDC)*.

Debian package   A package file format that originated with the Debian distribution but is now used on several other distributions. Debian packages feature excellent dependency tracking and easy installation and removal procedures.

default route   The route that network packets take if a more specific route doesn't direct them in some other way. The default route typically involves a gateway or router system that can further redirect the packets.

dependency   A requirement of one software package that another one be installed. For instance, most Linux programs include a dependency on the C library.

desktop environment   A set of programs that provide a friendly graphical environment for a Linux user.

development kernel   A kernel with an odd middle number, such as 2.5.67. These kernels incorporate experimental features and are not as stable as release kernels. See also *release kernel*.

DHCP   See *Dynamic Host Configuration Protocol (DHCP)*.

DHCP lease   A temporary assignment of an IP address to a DHCP client by a DHCP server. Clients must periodically renew their DHCP leases or risk losing the right to use the address.

digital subscriber line (DSL)   A type of broadband network access provided over telephone lines. Several subtypes of DSL exist, including Asymmetric DSL (ADSL) and Symmetric DSL (SDSL).

direct memory addressing (DMA)   A means of transferring data between devices (such as sound cards or SCSI host adapters) and memory without directly involving the CPU.

disk quota   A limit on the amount of disk space that an individual or group may use.

DMA   See *direct memory addressing (DMA)*.

DNS   See *Domain Name System (DNS)*.

Domain Name System (DNS)   A distributed set of computers that run servers to convert between computer names (such as `ns.example.com`) and IP addresses (such as 192.168.45.204). DNS servers are organized hierarchically and refer requests to systems responsible for successively more specific domains.

domain name   A name assigned to a group of computers, such as `example.com`. Individual computers have hostnames that include the domain name, such as `jupiter.example.com`.

dot file   A Linux or Unix file whose name begins with a dot (`.`). Most Linux shells and programs hide such files from the user, so user configuration files usually come in this form so as to be unobtrusive in directory listings.

dotted quad   A method of referring to an IP address or netmask that uses four 1-byte numbers separated by dots (`.`), as in 192.168.72.27 or 255.255.255.0.

DSL   See *digital subscriber line (DSL)*.

Dynamic Host Configuration Protocol (DHCP)   A protocol used on local networks for dissemination of network configuration information. A single DHCP server can maintain information for many DHCP clients, reducing overall configuration effort.

dynamic library   A type of library that's stored as a separate file from an executable program but that's loaded along with the main program file. Dynamic libraries save disk space and RAM compared to static libraries. See also *library* and *static library*.

E

EEPROM   See *electronically erasable programmable read-only memory (EEPROM)*.

EHCI   See *Enhanced Host Controller Interface (EHCI)*.

electronically erasable programmable read-only memory (EEPROM)   A type of data storage chip that retains data when power has been turned off but that can be erased and rewritten electronically. Frequently used to store a computer's or plug-in card's BIOS.

encryption key   A number that's used in conjunction with an algorithm to scramble data in a way that can be descrambled only with the use of the same or a related number.

Enhanced Host Controller Interface (EHCI)   A type of controller for USB 2.0 ports.

environment variable   A setting that's available to any program running in a session. Environment variables can define features such as the terminal type being used, the path to search for executable programs, and the location of an X server for GUI programs.

export   1. As a noun, a directory that's shared via the Network File System (NFS) server. 2. As a verb, the act of sharing a directory via NFS.

ext2   See *Second Extended File System (ext2fs or ext2)*.

ext2fs   See *Second Extended File System (ext2fs or ext2)*.

ext3   See *Third Extended File System (ext3fs or ext3)*.

ext3fs   See *Third Extended File System (ext3fs or ext3)*.

extended partition   A type of disk partition used on *x*86 systems. Extended partitions are placeholders for one or more logical partitions.

Extents File System (XFS)   One of several journaling filesystems for Linux. XFS was developed by Silicon Graphics (SGI) for its IRIX OS and then ported to Linux.

F

FAQ   See *Frequently Asked Question (FAQ)*.

FAT   See *File Allocation Table (FAT)*.

FHS   See *Filesystem Hierarchy Standard (FHS)*.

File Allocation Table (FAT)   1. A type of filesystem used as a native filesystem by DOS and Windows and also supported as a non-native filesystem by Linux and most other OSs. 2. On the FAT filesystem, a data structure after which the filesystem is named.

file globbing   The process of wildcard expansion—for instance, matching the existing file `glossary.txt` when the string `glos*.txt` is typed. Also called *globbing*.

file server   A computer or program that delivers files to other computers via network protocols upon request. Examples of file server programs include NFS, Samba, and FTP.

file type code   A special code that identifies the type of a file, such as a regular file, a directory, or a device file.

Filesystem Hierarchy Standard (FHS)   A standard that defines the names and contents of critical directories in a Linux filesystem (meaning 2).

Filesystem Standard (FSSTND)   An early attempt to define the names and contents of critical directories in a Linux filesystem (meaning 2). The FSSTND has been supplanted by the FHS.

filesystem   1. The low-level data structures recorded on a disk in order to direct the placement of file data. The filesystem determines characteristics like the maximum partition size, the file-naming rules, and what extra data (time stamps, ownership, and so on) may be associated with a file. 2. The overall layout of files and directories on a computer. For instance, a Linux filesystem includes a root directory (/), several directories falling off this (/usr, /var, /boot, etc.), subdirectories of these, and so on.

firewall   1. A program or kernel configuration that blocks access to specific ports or network programs on a computer. 2. A computer that's configured as a router and that includes firewall software that can restrict access between the networks it manages.

font server   A program that provides font bitmaps to client programs on the same or (sometimes) other computers. The font server may work directly from font bitmaps, or it may generate the bitmaps from outline fonts such as PostScript Type 1 or TrueType fonts.

font smoothing   See *anti-aliasing*.

frame   In networking, a data packet associated with network hardware (such as Ethernet) as opposed to the software (such as TCP/IP).

Frequently Asked Question (FAQ)   1. A question that's asked frequently, particularly on Usenet newsgroups or other online discussion forums. 2. A document that collects many FAQs (meaning 1) and their answers.

FSSTND   See *Filesystem Standard (FSSTND)*.

full duplex   A mode of communication in which data can be transferred in two directions at the same time.

function   In the context of programming or scripting, a section of code that can be called by name from other sections of code in order to perform some specific task or set of computations.

G

GID   See *group ID (GID)*.

gigabit Ethernet   A variety of Ethernet that can transfer 1,000 megabits (1 gigabit) per second.

glibc   See *GNU C library (glibc)*.

globbing   See *file globbing*.

GMT   See *Greenwich Mean Time (GMT)*.

GNOME   See *GNU Network Object Model Environment (GNOME)*.

GNU C library (glibc)   A specific type of C library used on Linux systems since the late 1990s. See also *C library (libc)*.

GNU Network Object Model Environment (GNOME)   A common desktop environment for Linux, headquartered at http://www.gnome.org.

Grand Unified Boot Loader (GRUB)   A popular boot loader for Linux. GRUB can boot a Linux kernel or redirect the boot process to another boot loader in a non-Linux partition, thus booting other OSs. Similar to the competing Linux Loader (LILO). See also *boot loader*.

graphical user interface (GUI)   A method of human/computer interaction characterized by a graphical display, a mouse to move a pointer around the screen, and the ability to perform actions by pointing at objects on the screen and clicking a mouse button.

Greenwich Mean Time (GMT)   The time in Greenwich, England, unadjusted for Daylight Saving Time. Linux systems use this time internally and adjust to local time by knowing the system's time zone. See also *Coordinated Universal Time (UTC)*.

group   A collection of users. Files are owned by a user and a group, and group members may be given access to files independent of the owner and all other users. This feature may be used to enhance collaborative abilities by giving members of a group read/write access to particular files while still excluding those who aren't members of the group. It can also be used by system administrators to control access to system files and resources.

group administrator   A person with administrative authority over a group. A group administrator can add or delete members from a group and perform similar administrative tasks.

group ID (GID)   A number associated with a particular group. Similar to a *user ID (UID)*.

GRUB   See *Grand Unified Boot Loader (GRUB)*.

GUI   See *graphical user interface (GUI)*.

H

hacker   1. An individual who is skilled at using or programming computers and who enjoys using these skills in constructive ways. Many Linux programmers consider themselves hackers in this sense of the term. 2. A cracker (see also *cracker*). This use of the term is more prevalent in the mass media, but it is frowned upon in the Linux community.

half-duplex   A type of data transmission in which data can be sent in only one direction at a time.

hard link   A directory entry for a file that has another directory entry. All hard links are equally valid ways of accessing a file, and all must be deleted in order to delete a file. See also *soft link*.

hardware address   A code that uniquely identifies a single network interface. This address is built into the device itself rather than assigned in Linux.

hardware clock   A clock that's built into *x*86 (and most other computers') hardware. The hardware clock maintains the time when the system is powered down. See also *software clock*.

hashes   An encryption method in which a file or string is encoded in a manner that cannot be reversed. Hashes are commonly used for password storage and as a more secure variant on checksums, among other things. See also *checksum*.

header   In e-mail, a line that contains information about a message's delivery path, sender, recipient, or other meta-information.

header file   File that contains interface definitions for software routines contained in a library. Program source code that uses a library must refer to the associated header files.

here document   A form of redirection, denoted by <<, which takes the following lines of input to be passed to a program as standard input. Most often used to pass fixed input to a program as standard input in scripts, obviating the need for separate support files.

HFS   See *Hierarchical File System (HFS)*.

Hierarchical File System (HFS)   A filesystem used on Mac OS.

high-level formatting   A type of disk formatting that writes the data that define a filesystem. Also called *making a filesystem*. See also *low-level formatting*.

hostname   A computer's human-readable name, such as `persephone.example.com`.

hot-pluggable   Hardware that can be safely added or removed while the computer is powered up and running. USB and FireWire are examples of protocols that support hot-plugging hardware.

HTTP   See *Hypertext Transfer Protocol (HTTP)*.

hub   A type of network hardware that serves as a central exchange point in a network. Each computer has a cable that links to the hub, so all data pass through the hub. Hubs echo all data they receive to all the other computers to which they connect. See also *switch*.

hung   Term used to describe a program that's stopped responding to user input, network requests, or other types of input to which it should respond. Hung processes sometimes consume a great deal of CPU time.

Hypertext Transfer Protocol (HTTP)   A protocol used for transferring web pages from a web server to a web browser.

I

ICMP   See *Internet Control Message Protocol (ICMP)*.

incremental backup   A type of backup in which only files that have changed since the last backup are backed up. This is used to reduce the time required to back up a computer, at the cost of potentially greater restoration complexity.

Industry Standard Architecture (ISA)   The expansion bus used on the original IBM PC. Most manufacturers began dropping ISA from their motherboards around 2001. ISA is inferior to PCI in most respects, but it has a huge installed base.

inode   A filesystem (meaning 1) data structure that contains critical information on the file, such as its size and location on the disk.

Integrated Services Digital Network (ISDN)   A type of digital telephone service and Internet access. ISDN supports a total data transfer speed of 128Kbps, plus some extra for control features.

Internet Control Message Protocol (ICMP)   A type of network packet that's commonly used to signal error conditions, such as corrupted packets.

Internet Printing Protocol (IPP)   A relatively new protocol for printing on a network. Used by CUPS on Linux.

Internet Protocol (IP)   An internet-layer protocol that's an important part of the TCP/IP network stack.

Internet Relay Chat (IRC)   An Internet-based real-time group communication system. IRC enables users to "chat" in forums devoted to particular topics. This can be handy for real-time debugging of Linux problems if you can find an appropriate forum.

interrupt request (IRQ)   A method by which peripherals (SCSI host adapters, sound cards, etc.) signal that they require attention from the CPU. An IRQ also refers to a specific interrupt signal line. The x86 architecture supports 16 IRQs, numbered 0–15, but IRQs 2 and 9 are linked, so in practice, there are only 15 IRQs, and many of these are used by basic hardware like floppy disks.

IP   See *Internet Protocol (IP)*.

IP address   A computer's numeric TCP/IP address, such as 192.168.45.203.

IP masquerading   See *Network Address Translation (NAT)*.

IPP   See *Internet Printing Protocol (IPP)*.

IPv6   The "next-generation" Internet Protocol. This upgrade to TCP/IP allows for a theoretical maximum of approximately 3.4×10^{38} addresses, as opposed to the 4 billion addresses possible with the IPv4 that's in common use in 2005.

IRC   See *Internet Relay Chat (IRC)*.

IRQ   See *interrupt request (IRQ)*.

ISA   See *Industry Standard Architecture (ISA)*.

ISDN   See *Integrated Services Digital Network (ISDN)*.

ISO-9660   The most common filesystem on CD-ROM and related optical media. ISO-9660 is often paired with *Rock Ridge extensions* or a *Joliet* filesystem in order to support long filenames and other features.

J

JFS   See *Journaled File System (JFS)*.

Joliet   A filesystem commonly used on CD-ROMs and related optical media. Joliet supports Microsoft-style long filenames and is almost always used in conjunction with an *ISO-9660* filesystem.

journal   An advanced filesystem feature that records data on pending disk operations. See also *journaling filesystem*.

Journaled File System (JFS)   One of several journaling filesystems for Linux. JFS was developed by IBM for its AIX OS. A subsequent implementation was created for OS/2, and Linux's JFS is derived from this code.

journaling filesystem   A type of filesystem that maintains a record of its operations. Such filesystems can typically recover quickly after a power failure or system crash. Common Linux journaling filesystems are ext3fs, ReiserFS, JFS, and XFS. See also *filesystem*.

K

K Desktop Environment (KDE)   A common desktop environment for Linux, headquartered at `http://www.kde.org`.

KDE   See *K Desktop Environment (KDE)*.

kernel module   A driver or other kernel-level program that may be loaded or unloaded as required.

kernel ring buffer   A record of recent messages generated by the Linux kernel. Immediately after a Linux system boots, this buffer contains the bootup messages generated by drivers and major kernel subsystems. This buffer may be viewed with the `dmesg` command.

L

LBA   See *logical block addressing (LBA)*.

LCD   See *liquid crystal display (LCD)*.

libc   See *C library (libc)*.

library   A collection of code that's potentially useful to many programs. This code is stored in special files to save disk space and RAM when running programs that use the library. See also *static library* and *dynamic library*.

LILO   See *Linux Loader (LILO)*.

linear block addressing (LBA)   See *logical block addressing (LBA)*.

link   A way of providing multiple names to reference a single file. Links are created with the `ln` command.

Linmodem   A software modem for which Linux drivers are available. See also *software modem*.

Linux Loader (LILO)   A popular Linux boot loader. Can boot a Linux kernel or redirect the boot process to another boot loader in a non-Linux partition, thus booting other OSs. Similar to the competing Grand Unified Boot Loader (GRUB). See also *boot loader*.

liquid crystal display (LCD)   A type of flat-panel display that's common on laptops and is becoming more common on desktop systems. LCDs are lightweight and consume little electricity, but they're more expensive to produce than are conventional monitors.

load average   A measure of the demands for CPU time by running programs. A load average of 0 means no demand for CPU time; 1 represents a single program placing constant demand on the CPU; and values higher than 1 represent multiple programs competing for CPU time. The `top` and `uptime` commands both provide load average information.

localhost   A name for the local computer.

log file   A text file, maintained by the system as a whole or by an individual server, in which important system events are recorded. Log files typically include information on user logins, server access attempts, and automatic routine maintenance.

log file rotation   A routine maintenance process in which the computer suspends recording data in log files, renames them, and opens new log files. This process keeps log files available for a time, but ultimately it deletes them, preventing them from growing to consume all available disk space. Also called *log rotation*.

log rotation   See *log file rotation*.

logical block addressing (LBA)   A method of accessing data on a disk that uses a single sector number to retrieve data from that sector. LBA contrasts with cylinder/head/sector (CHS) addressing. Some sources refer to LBA as *linear* block addressing.

logical partition   A type of *x*86 hard disk partition that has no entry in the primary partition table. Instead, logical partitions are carried within an extended partition.

loop   A programming or scripting construct enabling multiple executions of a segment of code. Typically terminated through the use of a conditional expression.

loopback   A name that refers to a network interface or address that points back to the computer itself. This is typically the 127.0.0.1 IP address.

low-level formatting   Creating data structures on a disk that define the locations of individual sectors and tracks. Hard disks are low-level formatted at the factory and should not normally be low-level formatted by end users. Floppy disks may need to be low-level formatted by end users via the Linux `fdformat` command. See also *high-level formatting*.

M

MAC address   See *Media Access Control (MAC) address.*

machine name   The portion of a hostname that identifies a computer on a network, as opposed to the network as a whole (for instance, `gingko` is the machine name portion of `gingkgo.example.com`). The machine name is sometimes used in reference to the entire hostname.

major version number   The first number in a program's version number. For instance, if a program's version number is 1.2.3, the major version number is 1.

master boot Record (MBR)   The first sector of a hard disk. The MBR contains code that the BIOS runs during the boot process as well as the primary partition table.

MBR   See *master boot record (MBR).*

Media Access Control (MAC) address   A low-level address associated with a piece of network hardware. The MAC address is usually stored on the hardware itself, and it is used for local network addressing only. Addressing between networks (such as on the Internet) uses higher-level addresses, such as an IP address.

mode   The permissions of a file. In conjunction with the file's owner and group, the mode determines who may access a file and in what ways.

mode lines   Definition of the timings required by particular video resolutions running at particular refresh rates.

modem   This word is short for *modulator/demodulator*. It's a device for transferring digital data over an analog transmission medium. Traditionally, the analog transmission medium has been the normal telephone network, but the word *modem* is increasingly being applied to devices used for broadband Internet access as well.

module   A kernel driver or other kernel component that's stored in a separate file. Linux can load modules on demand or on command, saving RAM when modules aren't in use and reducing the size of the kernel.

module stack   A set of modules that build up to provide some set of features. For instance, to deliver sound, you might need to load several sound driver modules that make up a module stack.

mount   1. The process of adding a filesystem (meaning 1) to a directory tree. 2. A command of the same name that performs this task.

mount point   A directory to which a new filesystem (meaning 1) is attached. Mount points are typically empty directories before their host filesystems are mounted.

mounted   The status of a filesystem that has been linked to a directory tree using the `mount` command.

multi-head display   A display that's made up of two or more physical monitors that together show a wider view on a larger virtual workspace.

multicast   A method of sending network data to multiple remote sites. Multicasts differ from broadcasts in that multicasts are more focused whereas broadcasts are typically sent to all the computers on a network.

N

NAT   See *Network Address Translation (NAT)*.

netmask   See *network mask*.

Network Address Translation (NAT)   A technique in which a router can "hide" a whole network from view, making all the systems look like one computer to the outside world.

Network File System (NFS)   A file sharing protocol used among Linux and Unix computers.

network mask   A bit pattern that identifies the portion of an IP address that's an entire network and the part that identifies a computer on that network. The pattern may be expressed in dotted quad notation (as in 255.255.255.0) or as the number of network bits following an IP address and a slash (as in 192.168.45.203/24). The network mask is also referred to as the *netmask* or *subnet mask*.

network stack   See *protocol stack*.

Network Time Protocol (NTP)   A network protocol and server enabling one computer to set its clock based on the value maintained by another clock.

New Technology File System (NTFS)   The favored filesystem on Windows NT/200*x*/XP systems. Linux supports NTFS, but this support is limited.

NFS   See *Network File System (NFS)*.

NTFS   See *New Technology File System (NTFS)*.

NTP   See *Network Time Protocol (NTP)*.

O

OHCI   See *Open Host Controller Interface (OHCI)*.

Open Host Controller Interface (OHCI)   One of two common hardware standards for managing USB 1.*x* ports. The other is the *Universal Host Controller Interface (UHCI)*.

open relay   An SMTP mail server that's configured to relay mail from anywhere to anywhere. Open relays are frequently abused by spammers to obfuscate their messages' true origins.

Open Sound System (OSS)   One of two common sound systems for Linux, the other being the *Advanced Linux Sound Architecture (ALSA)*. OSS is available in all versions of the Linux kernel, but ALSA is preferred in the 2.6.*x* kernel series.

OSS   See *Open Sound System (OSS)*.

outline font   See *scalable font*.

P

package   A collection of files stored in a single carrier file, ready for installation using a package management system such as RPM or the Debian package system.

packet   A limited amount of data collected together with addressing information and sent over a network.

packet filter firewall   A type of firewall that operates on individual network data packets, passing or rejecting packets based on information such as the source and destination addresses and ports.

packet sniffer   A program that monitors network traffic at a low level, enabling diagnosis of problems and capturing data. Packet sniffers can be used both for legitimate network diagnosis and for data theft.

parallel ATA (PATA)   The traditional form of ATA interface, in which several bits are transferred at once. See also *serial ATA (SATA)*.

PATA   See *parallel ATA (PATA)*.

path   A colon-delimited list of directories in which program files may be found. (Similar lists define the locations of directories, fonts, and other file types.)

PCI   See *Peripheral Component Interconnect (PCI)*.

PCL   See *Printer Control Language (PCL)*.

Peripheral Component Interconnect (PCI)   An expansion bus capable of much higher speeds than the older ISA bus. Modern computers usually include several PCI slots.

permission bit   A single bit used to define whether a given user or class of users has a particular type of access to a file. For instance, the owner's execute permission bit determines whether the owner can run a file as a program. The permission bits together make up the file's mode.

phishing   The process of sending bogus e-mail or putting up fake websites with the goal of collecting sensitive personal information (typically credit card numbers).

pipe   A method of executing two programs so that one program's output serves as the second program's input. Piped programs are separated in a Linux shell by a vertical bar (|).

pipeline   See *pipe*.

Plug and Play (PnP)   A set of hardware standards enabling automated, or at least software-based, configuration of hardware. This term applies mostly to ISA devices, although PCI and other devices are sometimes described as being PnP devices.

PnP   See *Plug and Play (PnP)*.

Point-to-Point Protocol (PPP)   A method of initiating a TCP/IP connection between two computers over an RS-232 serial line or modem. See also *PPP over Ethernet (PPPoE)*.

port   See *port number*.

port number   A number that identifies the program from which a data packet comes or to which it's addressed. When a program initiates a network connection, it associates itself with one or more ports, enabling other computers to uniquely address the program.

POST   See *power-on self-test (POST)*.

PostScript   A programming language used on many high-end printers. PostScript is optimized for displaying text and graphics on the printed page. The Linux program Ghostscript converts from PostScript to bitmapped formats understood by many low-end and mid-range printers.

PostScript Printer Definition (PPD)   A configuration file that provides information on a printer's capabilities—its paper size, whether it prints in color, and so on.

PostScript Type 1   An outline font format associated with PostScript but useable under Linux with or without a PostScript printer. See also *TrueType*.

power-on self-test (POST)   A series of basic hardware checks performed by the BIOS when the computer is first powered on and before the boot loader or OS boots. The POST can detect some, but not all, serious hardware problems before the OS boots.

PPD   See *PostScript Printer Definition (PPD)*.

PPP   See *Point-to-Point Protocol (PPP)*.

PPP over Ethernet (PPPoE)   A variant of the Point-to-Point Protocol (PPP) that's optimized for use over Ethernet rather than RS-232 serial connections. PPPoE is used by some DSL providers.

PPPoE   See *PPP over Ethernet (PPPoE)*.

primary partition   A type of x86 partition that's defined in a data structure contained in the hard disk's partition table in the MBR. An x86 computer can host only four primary partitions per hard disk.

print queue   A storage place for files waiting to be printed.

Printer Control Language (PCL)   A language developed by Hewlett-Packard for controlling printers. (Many of Hewlett-Packard's competitors now use PCL.) PCL is most commonly found on mid-range laser printers, but some ink-jet printers also support the language. Several PCL variants exist, the most common ranging from PCL 3 to PCL 6.

printer driver   A software component that converts printable data generated by an application into a format that's suitable for a specific model of printer. In Linux, printer drivers usually reside in Ghostscript, but some applications include a selection of printer drivers to print directly to various printers.

private key   One of two keys used in certain types of cryptography. The private key should be carefully guarded against theft. See also *encryption key* and *public key*.

privileged port   A port (see *port number*) that's numbered below 1024. Linux restricts access to such ports to `root`. In networking's early days, any program running on a privileged port could be considered trustworthy because only programs configured by professional system administrators could be run on such ports. Today, that's no longer the case. See also *unprivileged port*.

process   A piece of code that's maintained and run by the Linux kernel separately from other pieces of code. Most processes correspond to programs that are running. One program can be run multiple times, resulting in several processes.

protocol stack   A collection of drivers, kernel procedures, and other software that implements a standard means of communicating across a network. Two computers must support compatible protocol stacks to communicate. The most popular protocol stack today is TCP/IP. Also called a *network stack*.

public key   One of two keys used in certain types of cryptography. The public key is frequently given to the other side in a communication link. See also *encryption key* and *private key*.

R

random access   A method of access to a storage device (RAM, hard disk, etc.) by which information may be stored or retrieved in an arbitrary order with little or no speed penalty. See also *sequential access*.

RAM   See *random access memory (RAM)*.

random access memory (RAM)   Memory that can be randomly accessed. More specifically, RAM can be read and written with ease and makes up the bulk of the memory in modern computers.

real-time clock (RTC)   See *software clock*.

recursive lookup   A method of name resolution in which the DNS server queries a series of DNS servers, each of which has information on more and more specific networks, in order to locate the IP address associated with a hostname.

redirection   A procedure in which a program's standard output is sent to a file rather than to the screen or in which the program's standard input is obtained from a file rather than from the keyboard. See also *standard input* and *standard output*.

regular expression   A method of matching textual information that may vary in important ways but that contains commonalities. The regular expression captures the commonalities and uses various types of wildcards to match variable information.

ReiserFS   One of several journaling filesystems for Linux. ReiserFS was developed from scratch for Linux.

release kernel   A kernel with an even second number, such as 2.4.22 or 2.6.1. Release kernels should have few bugs, but they sometimes lack drivers for the latest hardware. See also *development kernel*.

release number   See *build number*.

Rock Ridge extensions   A set of extensions to the ISO-9660 filesystem that enable storage of Unix-style long filenames, ownership, permissions, and other filesystem features on an ISO-9660 filesystem. See also *ISO-9660* and *Joliet*.

root directory   The directory that forms the base of a Linux filesystem (meaning 2). All other directories are accessible from the root directory, either directly or via intermediate directories.

root filesystem   See *root directory*.

root kit   A set of scripts and other software that enable script kiddies to break into computers.

root servers   A set of DNS servers that deliver information to other DNS servers about top-level domains (`.com`, `.net`, `.us`, and so on). DNS servers consult the root DNS servers first when performing full recursive DNS lookups.

RPM   See *RPM Package Manager (RPM)*.

RPM Package Manager (RPM)   A package file format and associated utilities designed by Red Hat but now used on many other distributions as well. RPM features excellent dependency tracking and easy installation and removal procedures.

RTC   See *real-time clock (RTC)*.

runlevel   A number associated with a particular set of services that are being run. Changing runlevels changes services or can shut down or restart the computer.

S

Samba   A server that implements the SMB/CIFS file sharing protocols for Linux.

SAS   See *Serial Attached SCSI (SAS)*.

SATA   See *Serial ATA*.

scalable font   A font whose characters are defined in terms of a series of lines and curves. Scalable fonts are easily scaled to fit any display, printer, or other output device, but this scaling operation consumes CPU time. Also known as an *outline font*. See also *bitmap font*.

script kiddies   Individuals with little knowledge or skill who break into computers using scripts created by others. Such break-ins often leave obvious traces, and script kiddies frequently cause collateral damage that produces system instability.

SCSI   See *Small Computer System Interface (SCSI)*.

Second Extended File System (ext2fs or ext2)   The most common filesystem (meaning 1) in Linux from the mid-1990s through approximately 2001.

Secure Shell (SSH)   A remote login protocol and program that uses encryption to ensure that intercepted data packets cannot be used by an interloper. Generally regarded as the successor to Telnet on Linux systems.

sendmail   A popular SMTP mail server for Linux.

sequential access   A method of accessing a storage medium; sequential access requires reading or writing data in a specific order. The most common example is a tape; to read data at the end of a tape, you must wind past the interceding data. See also *random access*.

Serial ATA (SATA)   A type of ATA interface that uses serial data transfer rather than the parallel data transfers used in older forms of ATA. See also *parallel ATA (PATA)*.

Serial Attached SCSI (SAS)   A type of SCSI interface that uses serial data transfer rather than the parallel data transfers used in older forms of SCSI.

Serial Line Internet Protocol (SLIP)   A method of initiating a TCP/IP connection between two computers over an RS-232 serial line or modem. Although once common, SLIP is seldom used today; instead, the *Point-to-Point Protocol (PPP)* is used.

set group ID (SGID)   A special type of file permission used on program files to make the program run with the permissions of its group. (Normally, the user's group permissions are used.)

set user ID (SUID)   A special type of file permission used on program files to make the program run with the permissions of its owner rather than those of the user who runs the program.

SGID   See *set group ID (SGID)*.

shareable files   Files that can be reasonably shared with another computer, as in users' home directory files and program files in /opt or /usr.

shared library   See *dynamic library*.

shell   A program that provides users with the ability to run programs, manipulate files, and so on.

shell history   A log of commands typed at a shell. The shell history enables easy repetition of a previously typed command.

shell script   A program written in a language that's built into a shell.

signal   In reference to processes, a signal is a code that the kernel uses to control the termination of the process or to tell it to perform some task. Signals can be used to kill processes.

Simple Mail Transfer Protocol (SMTP)   The most common push mail protocol on the Internet. SMTP is implemented in Linux by servers such as sendmail, Postfix, Exim, and qmail.

skeleton directory   A directory, typically `/etc/skel`, that holds files that should be copied to each new user's home directory as the user's account is created.

SLIP   See *Serial Line Internet Protocol (SLIP)*.

Small Computer System Interface (SCSI)   An interface standard for hard disks, CD-ROM drives, tape drives, scanners, and other devices.

smart filter   A program, run as part of a print queue, that determines the type of a file and passes it through appropriate programs to convert it to a format that the printer can handle.

smart relay   A mail server configuration that involves sending all outgoing mail to a specific upstream mail server rather than directly to the destination system. Such configurations are common on small networks in which the ISP blocks direct SMTP connections to any but its own mail server as an anti-spam measure or when you want one of your own systems to handle all of a network's outgoing mail as a tracking measure.

SMTP   See *Simple Mail Transfer Protocol (SMTP)*.

social engineering   The practice of convincing individuals to disclose sensitive information without arousing suspicion. Social engineers may pretend to be system administrators to ask for passwords, for instance. See also *phishing*.

socket   A programming construct enabling connection to network connection endpoints (the combination of IP addresses and port numbers).

soft link   A type of file that refers to another file on the computer. When a program tries to access a soft link, Linux passes the contents of the linked-to file to the program. If the linked-to program is deleted, the soft link stops working. Deleting the soft link doesn't affect the original file. Also referred to as a *symbolic link*. See also *hard link*.

software clock   A type of clock maintained by a running Linux system and used by most software programs that must refer to the time. See also *hardware clock*.

software modem   A modem that implements key functionality in software that must be run by the host computer. These modems require special drivers, which are uncommon in Linux.

spool directory   A directory in which print jobs, mail, or other files wait to be processed. Spool directories are maintained by specific programs, such as the printing system or SMTP mail server.

squashing   In the context of the NFS file server, this refers to changing the effective UID or GID of an access request to limit the system's vulnerability. The `root` account on the client is often squashed to the `nobody` account on the server.

SSH   See *Secure Shell (SSH)*.

stable kernel   See *release kernel*.

standard error   An output stream that's reserved for high-priority messages, such as errors. See also *standard output*.

standard input   The default method of delivering input to a program. It normally corresponds to the keyboard at which you type.

standard output   The default method of delivering purely text-based information from a program to the user. It normally corresponds to a text-mode screen, `xterm` window, or the like. See also *standard error*.

stateful packet inspection   A firewall tool in which a packet's state (that is, whether it's marked to begin a transaction, to continue an existing exchange, and so on) is considered in the filtering process.

static file   A file that seldom changes. Partitions that hold nothing but static files may be mounted read-only to minimize the risk of accidental or malicious changes to the partition's files.

static library   A type of library that's compiled into the program's main executable file. This contrasts with a *dynamic library*.

sticky bit   A special file permission bit that's most commonly used on directories. When set, only a file's owner may delete the file, even if the directory in which it resides can be modified by others.

stratum   In the context of the Network Time Protocol (NTP), the distance of a server from an atomic clock or other original time source. Stratum 0 servers are such sources but cannot be contacted directly except by a stratum 1 server. Stratum 2 servers set their clocks from stratum 1 servers, stratum 3 servers set their clocks from stratum 2 servers, and so on.

stream   Text or other input or output as processed by a program. Examples include files, keyboard input, and output to a screen.

subdomain   A subdivision of a domain. A subdomain may contain computers or subdomains of its own.

subnet mask   See *network mask*.

SUID   See *set user ID (SUID)*.

super server   A server that listens for network connections intended for other servers and launches those servers. Examples on Linux are `inetd` and `xinetd`.

superuser   A user with extraordinary rights to manipulate critical files on the computer. The superuser's username is normally `root`.

swap file   A disk file configured to be used as swap space.

swap partition   A disk partition configured to be used as swap space.

swap space   Disk space used as an extension to a computer's RAM. Swap space enables a system to run more programs or to process larger data sets than would otherwise be possible.

switch   A type of network hardware that serves as a central exchange point in a network. Each computer has a cable that links to the switch, so all data pass through the switch. A switch usually sends data only to the computer to which it's addressed. See also *hub*.

symbolic link   See *soft link*.

system clock   See *hardware clock*.

system cron job   A cron job that handles system-wide maintenance tasks, like log rotation or deletion of unused files from /tmp. See also *user cron job*.

System V (SysV)   A form of AT&T Unix that defined many of the standards used on modern Unix systems and Unix clones, such as Linux.

SysV   See *System V (SysV)*.

SysV startup script   A type of startup script that follows the System V startup standards. Such a script starts one service or related set of services.

T

tarball   A package file format based on the tar utility. Tarballs are easy to create and are readable on any version of Linux and on most non-Linux systems. They contain no dependency information and the files they contain are not easy to remove once installed, however.

TCP   See *Transmission Control Protocol (TCP)*.

TCP/IP   See *Transmission Control Protocol/Internet Protocol (TCP/IP)*.

terminal program   A program that's used to initiate a simple text-mode connection between two computers, especially via a modem or RS-232 serial connection.

terminated   In the context of SCSI devices, this refers to the presence of a resistor pack that prevents signals from bouncing back from the end of a SCSI chain. The devices on both ends of a SCSI chain must be terminated, but other devices must not be terminated.

Third Extended File System (ext3fs or ext3)   A variant of the *Second Extended Filesystem (ext2 or ext2fs)* that adds a journal to reduce startup times after a power failure or system crash. See also *journaling filesystem*.

Transmission Control Protocol (TCP)   A major transport-layer protocol type in modern networking. TCP supports error correction and other features that are helpful in maintaining a link between two computers.

Transmission Control Protocol/Internet Protocol (TCP/IP)   The most important protocol stack in common use today, and the basis for the Internet.

TrueType A font format developed by Apple and dominant in the Mac OS and Windows worlds. TrueType support in Linux is provided by certain font servers, by XFree86 4.*x*, by X.org-X11, and by Xfs. See also *PostScript Type 1*.

trusted hosts A security system in which the server trusts a specified set of clients to manage key aspects of the security. The trusted hosts model is used by NFS and some other Linux servers but is risky on the Internet at large.

tunnel The process of using one protocol to carry another protocol's traffic. Tunneling enables you to gain the advantages of one protocol when using another. For instance, the Secure Shell (SSH) may be used to tunnel various other protocols, thus adding encryption to a non-encrypted protocol.

twisted-pair A type of wiring in which pairs of wires are wrapped around each other (and typically enclosed in a plastic sheath). Twisted-pair cabling is commonly used for Ethernet and certain other network cabling. Many telephone wires also use twisted-pair cabling.

U

UDF See *Universal Disc Format (UDF)*.

UDP See *User Datagram Protocol (UDP)*.

UHCI See *Universal Host Controller Interface (UHCI)*.

UID See *user ID (UID)*.

umask See *user mask (umask)*.

Universal Disc Format (UDF) A next-generation optical disc filesystem, frequently used on DVD-ROMs.

Universal Host Controller Interface (UHCI) One of two common hardware standards for managing USB 1.*x* ports. The other is the *Open Host Controller Interface (OHCI)*.

Universal Serial Bus (USB) A type of interface for low- to medium-speed external devices, such as keyboards, mice, cameras, modems, scanners, and removable disk drives. Linux added USB support with the 2.2.18 and 2.4.*x* kernels. USB 2.0 increases the speed to the point that USB is useable for hard disks in less-demanding applications.

unprivileged port A port (see *port number*) that's numbered above 1024. Such ports may be accessed by any user and so are commonly used by client programs and by a few servers that may legitimately be run by ordinary users. See also *privileged port*.

unshareable file A file that should not be shared with other computers. Examples include system-specific configuration files (such as most of /etc) and system-specific queue directories in certain subdirectories of /var.

USB   See *Universal Serial Bus (USB)*.

USB hub   A piece of hardware that enables connecting multiple USB devices to a single USB connector on a computer.

user `cron` **job**   A `cron` job created by an individual user to handle tasks for that user, such as running a CPU-intensive job late at night when other users won't be disturbed by the job's CPU demands. See also *system* `cron` *job*.

User Datagram Protocol (UDP)   A transport-layer protocol used on the TCP/IP stack. UDP is a simple and efficient protocol, but it lacks error checking and other advanced features that can be helpful in maintaining a connection.

user ID (UID)   A number associated with a particular account. Linux uses the UID internally for most operations, and it converts to the associated username only when interacting with people.

user mask (umask)   A bit pattern representing the permission bits that are to be removed from files created from a process.

username   The name associated with an account, such as `theo` or `miranda`. Linux usernames are case sensitive and may be from 1 to 32 characters in length, although they're usually entirely lowercase and no longer than 8 characters.

UTC   See *Coordinated Universal Time (UTC)*.

V

variable   In computer programming or scripting, a "placeholder" for data. Variables may change from one run of a program to another, or even during a single run of a program.

variable file   A file whose data may change at any time. Examples include user data files and queues maintained by servers.

virtual filesystem   A filesystem that doesn't correspond to a real disk partition, removable disk, or network export. A common example is `/proc`, which provides access to information on the computer's hardware.

W

widget set   A collection of programming tools that help programmers create GUI displays, such as dialog boxes, menus, and scroll bars.

widget   A control or decoration in a GUI display, such as a button or menu.

wildcard   A character or group of characters that, when used in a shell as part of a filename, match more than one character. For instance, b??k matches book, back, and buck, among many other possibilities.

window manager   A program that provides decorative and functional additions to the plain windows provided by X. Linux supports dozens of window managers.

Winmodem   See *software modem.*

X

X   See *X Window System.*

X.org-X11   The most common X server for Linux. Derived from XFree86.

X core fonts   Fonts managed by the X server.

X Display Manager Control Protocol (XDMCP)   A protocol for managing X connections across a network. XDMCP is also used for displaying a local GUI login screen for Linux workstations. Common XDMCP servers for Linux include the X Display Manager (XDM), the GNOME Display Manager (GDM), and the KDE Display Manager (KDM).

X logical font description (XLFD)   A format for describing X core fonts. Programs use the XLFD when telling X to display a font, and the XLFD also appears in X core font configuration files.

X server   A program that implements X for a computer; especially the component that interacts most directly with the video hardware.

X Window System   The GUI environment for Linux. The X Window System is a network-aware, cross-platform GUI that relies on several additional components (such as a window manager and widget sets) to provide a complete GUI experience. See also *X server, XFree86,* and *X.org-X11.*

XDMCP   See *X Display Manager Control Protocol (XDMCP).*

XFree86   A set of X servers and related utilities for Linux and other OSs. Abandoned on most distributions in favor of X.org-X11.

XFS   See *Extents File System (XFS).*

XLFD   See *X logical font description (XLFD).*