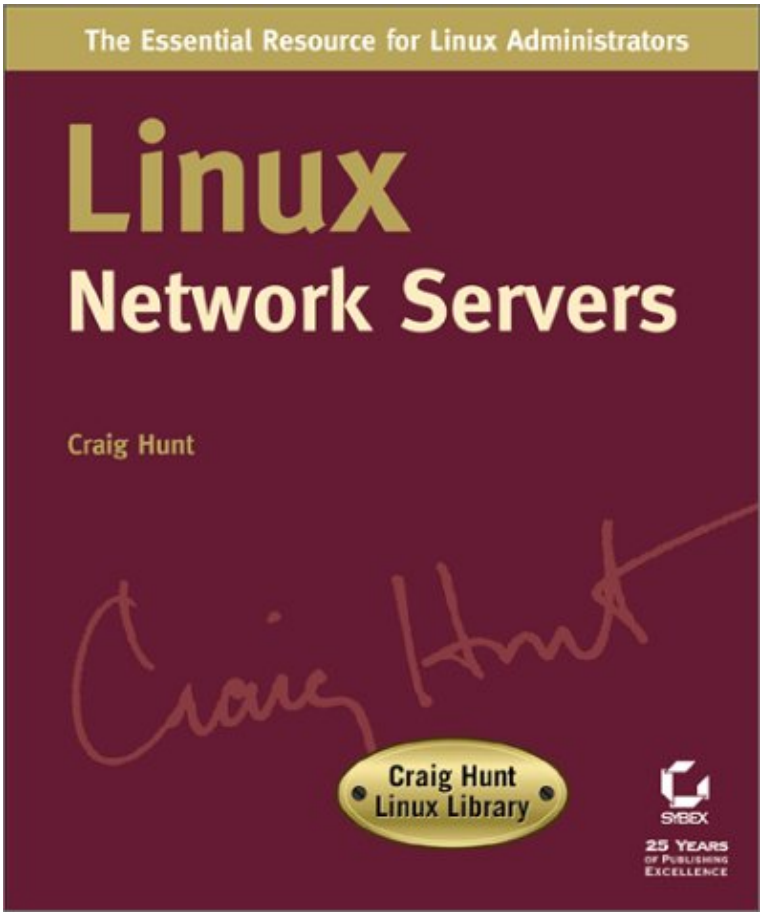


Linux Network Servers

Table of Contents

Linux Network Servers **1**

Foreword **2**
Acknowledgments 2

Introduction **4**
Who Should Buy This Book 5
How This Book Is Organized 5
Part 1: The Basics 6
Chapter 1: The Boot Process 6
Chapter 2: The Network Interface 6
Part 2: Internet Server Configuration 6
Chapter 3: Login Services 6
Chapter 4: Linux Name Services 6
Chapter 5: Configuring a Mail Server 7
Chapter 6: The Apache Web Server 7
Chapter 7: Network Gateway Services 7
Part 3: Departmental Server Configuration 7
Chapter 8: Desktop Configuration Servers 7
Chapter 9: File Sharing 7
Chapter 10: Printer Services 7
Chapter 11: More Mail Services 8
Part 4: Maintaining a Healthy Server 8
Chapter 12: Security 8
Chapter 13: Troubleshooting 8
Part 5: Appendices 8
Appendix A: Installing Linux 8
Appendix B: BIND Reference 8
Appendix C: The m4 Macros for sendmail 9
Conventions 9
Help Us Help You 10

Part I: The Basics **11**
Chapter List 11
Part Overview 11
Featuring: 11

Chapter 1: The Boot Process **12**
Overview 12
Loading the Boot Sector 12
Loading Linux with GRUB 14
Loading the Kernel with LILO 17
LILO Configuration Options 17
The Linux Boot Prompt 21
Hardware Device Driver Initialization 24
Loading Linux Services—The init Process 25
Understanding Runlevels 26
Special–Purpose Entries 28
Startup Scripts 29
System Initialization 29

Table of Contents

Chapter 1: The Boot Process
Runlevel Initialization....30
Controlling Scripts....31
The rc.local Script....33
Loadable Modules....33
Listing the Loaded Modules....33
In Sum....35

Chapter 2: The Network Interface....36
Overview....36
Configuring an Ethernet Interface....36
Loadable Ethernet Drivers....36
The ifconfig Command....39
Network Interface Configuration Tools....42
The Serial Interface....43
Connecting through the Serial Interface....44
Running TCP/IP Over a Serial Port....46
Installing PPP....46
The PPP Kernel Module....47
The PPP Daemon....48
Configuring a PPP Server....49
PPP Dial−Up Server Configuration....49
PPP Security....51
PPP Client Configuration....53
chat Scripts....54
Using an X Tool to Configure a PPP Client....55
In Sum....57

Part II: Internet Server Configuration....58
Chapter List....58
Part Overview....58
Featuring:....58

Chapter 3: Login Services....59
Overview....59
Starting Services On−Demand....60
Protocol and Port Numbers....60
Configuring inetd....63
Configuring xinetd....65
Creating User Accounts....70
The Steps to Creating a User Account....70
The passwd File....70
Tools to Create User Accounts....75
Additional FTP Configuration....80
The ftpaccess File....82
In Sum....84

Chapter 4: Linux Name Services....86
Overview....86
The hosts File....86

Table of Contents

Chapter 4: Linux Name Services
Understanding DNS....87
The DNS Hierarchy....87
Answering Queries....88
The BIND Software....88
Configuring the Resolver....89
The Lightweight Resolver....94
Configuring a Domain Name Server....96
The named Configuration File....97
A Caching–Only Configuration....101
The Slave Server Configuration....106
The Master Server Configuration....107
Running named....119
named Signal Processing....120
The named Control Tools....121
Using the Host Table with DNS....124
In Sum....127

Chapter 5: Configuring a Mail Server....128
Overview....128
Using Mail Aliases....128
Defining Personal Mail Aliases....131
Using sendmail to Receive Mail....131
The sendmail Configuration File....132
The Local Info Section....133
The Options Section....134
The Message Precedence Section....135
The Trusted Users Section....135
The Format of Headers Section....136
The Rewriting Rules Section....137
The Mailer Definitions Section....139
Configuring the sendmail.cf File....142
Testing Your New Configuration....143
Using m4 to Configure sendmail....145
The m4 Macro Control File....146
The Linux OSTYPE File....147
Creating an m4 DOMAIN File....148
Building the m4 Configuration File....151
Building a sendmail Database....152
Testing the m4 Configuration....152
In Sum....153

Chapter 6: The Apache Web Server....154
Overview....154
Installing Apache....154
Running httpd....156
Configuring the Apache Server....158
The httpd.conf File....159
Loading Dynamic Shared Objects....161
Basic Server Directives....163

Table of Contents

Chapter 6: The Apache Web Server
Defining Where Things Are Stored....165
Creating a Fancy Index....166
Defining File Types....167
Managing Child Processes....167
Performance Tuning Directives....169
Caching Directives....169
Defining Virtual Hosts....170
Web Server Security....171
The CGI and SSI Threat....172
Server Options for Documents and Directories....172
Directory–Level Configuration Controls....174
Defining Access Controls....175
Requiring User Authentication....177
Configuring SSL....179
Managing Your Web Server....186
Monitoring Your Server....187
Apache Logging....188
In Sum....191

Chapter 7: Network Gateway Services....192
Overview....192
Understanding Routing....194
Converting IP Addresses to Ethernet Addresses....194
Enabling IP Packet Forwarding....196
The Linux Routing Table....197
Defining Static Routes....199
The route Command....200
Using Dynamic Routing....201
Routing Protocols....201
Running RIP with routed....204
Routing with Zebra....206
Using gated....218
Network Address Translation....225
Configuring a Linux NAT Server....226
In Sum....227

Part III: Departmental Server Configuration....228
Chapter List....228
Part Overview....228
Featuring:....228

Chapter 8: Desktop Configuration Servers....229
Overview....229
Understanding Configuration Protocols....229
Bootstrap Protocol....229
Dynamic Host Configuration Protocol....230
Reverse Address Resolution Protocol....231
Installing the DHCP Server....231
Running dhcpd....233

Table of Contents

Chapter 8: Desktop Configuration Servers
Initializing the dhcpd.leases File....234
Configuring the DHCP Server....235
Controlling Server and Protocol Operations....235
dhcpd Configuration Options....237
Creating a dhcpd.conf File....242
Configuring a dhcrelay Server....243
Configuring a DHCP Client....246
Using the dhcpcd Client....246
Using the pump DHCP Client....249
Running dhclient Software....251
In Sum....255

Chapter 9: File Sharing....256
Overview....256
Linux Filesystem....256
Linux File Permissions....256
Changing File Permissions....258
The chgrp Command....260
Understanding NFS....260
Installing NFS....262
Configuring an NFS Server....264
Mapping User IDs and Group IDs....265
The exportfs Command....267
Configuring an NFS Client....268
The mount Command....269
The umount Command....270
Using fstab to Mount NFS Directories....270
Automounter....274
Understanding SMB and NetBIOS....276
NetBIOS Name Service....277
Installing Samba....279
Configuring a Samba Server....280
The smb.conf Variables....281
The smb.conf Global Section....282
The smb.conf Homes Section....284
Sharing a Directory through Samba....285
Using a Linux Samba Client....286
Using smbclient....287
Using smbmount....287
In Sum....289

Chapter 10: Printer Services....290
Installing Printers....290
Configuring Remote Printers....295
Understanding printcap....297
printcap Parameters....298
A Sample printcap....298
Sharing Printers with lpd....300
Using lpr....301

Table of Contents

Chapter 10: Printer Services
Managing lpd....301
Sharing Printers with Samba....304
Defining Printers in the smb.conf File....304
Printers Share Section....305
smb.conf Printer Configuration Options....306
Using an SMB Printer....306
In Sum....308

Chapter 11: More Mail Services....309
Overview....309
Understanding POP and IMAP....309
The POP Protocol....309
The IMAP Protocol....311
Running the POP and IMAP Daemons....314
Using POP or IMAP from a Client....315
Stopping Spam E–Mail....316
Don't Be a Spam Source....317
Using sendmail to Block Spam....319
Filtering Out Spam at the Mailer....324
In Sum....331

Part IV: Maintaining a Healthy Server....332
Chapter List....332
Part Overview....332
Featuring:....332

Chapter 12: Security....333
Overview....333
Understanding the Threats....333
The Basic Threats....333
A Reality Check....334
Keeping Informed....335
Closing the Holes....337
Finding the Latest Software....337
Removing Unneeded Software....339
Controlling Access with tcpd....340
Tracking Remote Access....341
tcpd Access Control Files....342
Controlling Network Access with xinetd....347
Controlling Access with iptables....350
Maintaining Firewall Rules with iptables....350
Sample iptables Commands....352
Improving Authentication....353
Shadow Passwords....354
One–Time Passwords....357
Secure Shell....359
Monitoring Your System....370
Security Monitoring Tools....370
In Sum....371

Table of Contents

Chapter 13: Troubleshooting **372**
Overview 372
Configuring the Linux Kernel 372
Configuring the Kernel with xconfig 373
Compiling and Installing the Kernel 377
Troubleshooting a Network Server 378
Diagnostic Tools 379
Checking the Network Interface 380
Checking an Ethernet Interface 381
Resolving Address Conflicts 384
Checking a PPP Interface 388
Testing the Connection 390
The Message of a Successful ping 390
The Message of a Failed ping 391
Testing Routing 392
Using traceroute 392
Analyzing Network Protocols 394
Checking Socket Status with netstat 394
Watching the Protocols with tcpdump 397
Testing Services 399
Testing DNS with nslookup 400
Testing DNS with host 402
Testing DNS with dig 403
In Sum 404

Appendices **405**
Appendix List 405

Appendix A: Installing Linux **406**
Overview 406
Installation Planning 407
Hardware Information 407
Network Information 408
Software Considerations 409
Selecting an Installation Method 409
Making a Boot Disk 410
Booting the Installation Program 411
Partitioning the Disk 413
Partition Planning 414
Partitioning with Disk Druid 417
Partitioning with fdisk 421
Installing the Boot Loader 424
Configuring the Ethernet Adapter 425
Configuring the Firewall 426
Installing the Software 429
X Windows 429
The Boot Floppy 431
In Sum 432

Table of Contents

Appendix B: BIND Reference **433**
Overview 433
named.conf Commands 433
The options Statement 433
The logging Statement 440
The zone Statement 442
The server Statement 445
The key Statement 446
The acl Statement 447
The trusted–keys Statement 447
The controls Statement 448
BIND 9 view Statement 449

Appendix C: The m4 Macros for sendmail **450**
Overview 450
define 452
FEATURE 461
OSTYPE 465
DOMAIN 467
MAILER 470
Local Code 471
DAEMON_OPTIONS 472
LDAP Mail Routing 473

List of Figures **474**

List of Tables **476**

List of Listings **478**

List of Sidebars **483**

Linux Network Servers

Craig Hunt

Associate Publisher: Neil Edde
Acquisitions and Developmental Editor: Maureen Adams
Editor: Nancy Sixsmith
Production Editor: Kylie Johnston
Technical Editor: Matthew Miller
Book Designer: Bill Gibson
Graphic Illustrator: Tony Jonick
Electronic Publishing Specialists: Judy Fung, Nila Nichols
Proofreaders: Dave Nash, Laurie O'Connell, Nancy Riddiough
Indexer: Ted Laux
Cover Designer: Ingalls & Associates
Cover Illustrator/Photographer: Ingalls & Associates

Copyright © 2002 SYBEX Inc., 1151 Marina Village Parkway, Alameda, CA 94501. World rights reserved. No part of this publication may be stored in a retrieval system, transmitted, or reproduced in any way, including but not limited to photocopy, photograph, magnetic, or other record, without the prior agreement and written permission of the publisher.

An earlier version of this book was published under the title *Linux Network Servers 24seven* © 1999 SYBEX Inc.

Library of Congress Card Number: 2002104868

ISBN: 0–7821–4123–4

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

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

To Norman Hunt and Frank McCafferty,
they showed me what it means to be a man.

Foreword

The Craig Hunt Linux Library is a series of technical books dedicated to providing professional Linux system administrators with the information they need to do a tough job effectively. The goal of the library is to provide highly technical books that are clear, accurate, and complete. The library currently includes eight titles, with *Linux Network Servers* being the latest addition. Most of the books in this series focus in great depth on a single subject, and a glance at titles such as *Linux Apache Web Server Administration* and *Linux DNS Server Administration* shows that most of the books in the Craig Hunt Linux Library focus on network services.

No matter what your involvement in networking, the Craig Hunt Linux Library has the right book for you. Starting with *Linux System Administration*, which has one chapter on TCP/IP networking, through *Linux Network Servers*, which has one chapter on each networking topic, to books such as *Linux Sendmail Administration* that dedicate an entire book to a single network topic, the level of detail that you need is provided by the books in this library.

The important roles that Linux plays supporting network services is not only obvious from the titles of books in this library, it is clear from industry reports that show the strong and growing role of Linux as a network server. The partnership of Apache and Linux has long been acknowledged by professional web masters, but the range of network service provided by Linux goes far beyond support for the leading web server software. Linux provides a full range of network services, and *Linux Network Servers* covers them all.

I am very pleased that *Linux Network Servers* has now become part of the Craig Hunt Linux Library. This book fits nicely into the mission of this library, rounds out the selection of titles, and adds a book of highly acclaimed quality. If you know Linux, you know Alan Cox. For the uninitiated, Alan Cox is the person that the *Linux Journal* called "the Linux community's own Mr. Wizard." In his review of a previous version of this book, he said:

"If I had to pick a reference book for a new Linux administrator or to have as a reference guide to Linux administration in the office, this would be it."

Enough said!

Craig Hunt

August 2002

Acknowledgments

This book again brought together Neil Edde and Maureen Adams, the team that first introduced me to Sybex. Neil, who is associate publisher for the Craig Hunt Linux Library, first proposed the idea of adding this book to the library. Maureen Adams, as the acquisitions editor for this series, got me pointed in the right direction and gave me the kick–start needed to get this book underway. Both of these fine people have my thanks.

The production editor for this book was Kylie Johnston. Kylie deserves special thanks for her ability to keep the project on schedule without alienating anyone. Nancy Sixsmith was the editor. I want to thank her for a light touch that improved the text without compromising my writing style. Matthew Miller was the technical editor. His suggestions were very helpful in creating a more accurate book.

The Sybex production team are consummate professionals. Thanks to Judy Fung and Nila Nichols, the compositors; Amey Garber, Dave Nash, Laurie O'Connell, and Nancy Riddiough, the proofreaders; Tony Jonick, the illustrator; and Ted Laux, the indexer.

I'd also like to thank Karen Ruckman of KJR Design in Washington D.C. Karen is a professional photographer and designer. I can attest to the fact that she is one of the best. Only the best of photographers could make my mug look presentable enough for the cover of a book.

Life can be very busy and complicated, yet deadlines remain unyielding and pressure builds. Thanks to Kathy, Sara, David, and Rebecca for enduring and diverting me. And a special thanks to little Alana for interrupting me with a charming smile when I didn't even know I needed to be interrupted.

Introduction

Linux is the perfect choice for an operating system on which to build a network server. Much of the fame of Linux as a server system comes from its widespread use as a system on which Apache web servers are built. But the power and reliability of Linux does more than provide a stable platform for the world's most popular web server. Linux provides all of the most important network services in a single low−cost package.

Low cost, reliability, and power are propelling the continued growth of Linux as a server system. Linux has proven to be a cost−effective alternative to high−cost Unix servers. And it has proven itself to be more powerful and reliable than any proprietary desktop operating system trying to recast itself as a server operating system. Sales people might lust after the vast desktop market, but as professional system administrators, we know that the real technical action is with the server systems.

The tremendous range of network services provided by Linux means that it can be used for all of your network server needs. In this book, servers are categorized as "Internet servers" and "departmental servers." This somewhat arbitrary division is done to organize the discussion of the various services in a rational way. We define Internet services as those services that are often offered to the world at large or that are used to connect an organization to the worldwide Internet. The services that are covered in this category are:

- Domain Name System (DNS) services
- sendmail
- Apache
- Login services such as FTP, Telnet, and SSH
- Routing protocols through Zebra and gated
- Network Address Translation (NAT)

Departmental services are those services that are usually limited to usage on the internal network. The services that are covered under this category are:

- Dynamic Host Configuration Protocol (DHCP)
- Reverse Address Resolution Protocol (RARP)
- Network File System (NFS)
- Samba file and printer sharing
- LPR/LPD printer sharing
- Post Office Protocol (POP)
- Internet Message Access Protocol (IMAP)
- procmail mail filtering

In addition to these specific topics, this book contains general information on configuring network interfaces, and important chapters on security and troubleshooting.

Linux Network Servers grew out of my earlier book, *Linux Network Servers 24seven*. This new book, however, is more than a second edition. Although the character and content that drew high praise for the original book remains, the new book has been completely reworked for the professional system administrators who rely on the Craig Hunt Linux Library. (Much of the praise for *Linux Network Servers 24seven* is still available online for your perusal.) Introductory material from the original book was removed to make room for more technical details in this version. I believe, and I hope you agree, that this new book is even better than its predecessor.

Who Should Buy This Book

You should! *Linux Network Servers* is for anyone who wants to learn how to build a departmental server or an Internet server using Linux. The book doesn't assume that you know everything about Linux. But it does assume that you have a good understanding of computers and IP networks, and a basic understanding of Linux commands and Linux system administration. If you feel that you need to brush up on these topics, start with *Linux System Administration* (Stanfield and Smith, Sybex, 2002). It is an excellent introduction to Linux system administration, and will give Linux users all the background they need. If you're coming to Linux from a Windows NT background, you may want to start with *Linux for Windows NT/ 2000 Administrators* (Minasi, York, and Hunt, Sybex, 2000).

Linux Network Servers does not provide yet another review of the basics. Instead, it provides insight into how to get network service up and running quickly with information designed for professional system administrators.

Linux growth is making sharp inroads into the currently installed base of Unix servers. If you're a Unix professional retraining for a job as a Linux system administrator, this book is for you. You'll benefit from the detailed information on Linux–specific commands. Additionally you'll be pleased by the tremendous similarity between the two systems. This book may be all the information you need to move from Unix to Linux.

Linux system administrators will find this book invaluable as their primary resource for information on network services. Even administrators of servers dedicated to specific tasks, such as web servers or DNS servers, will find this book a useful companion text. Although such an administrator may rely on *Linux Apache Web Server Administration* or *Linux DNS Server Administration* as a primary resource, this book provides the insights into how other services work and how they are configured, which are helpful to anyone running a Linux server.

This book is not simply a reference to network server configuration options. Instead, it provides insight into how real servers are actually configured. This book helps you understand how things really work so that you can make intelligent configuration decisions that relate to your environment. No book, no matter how well–thought–out or how long, can provide accurate examples for every possible situation. This book strives to provide you with the information you need to develop the correct solution for your situation on your own.

How This Book Is Organized

Although this book is intended to be read as a whole, I understand that many system administrators simply do not have the time to read an entire text. They must go to the topic in question and get a reasonably complete picture of the "why" as well as the "how" of that topic. To facilitate that understanding, necessary background material is summar–ized where the topic is discussed, and it is accompanied by pointers to the part of the text where the background material is more thoroughly discussed.

This book is divided into five parts: The Basics, Internet Server Configuration, Departmental Server Configuration, Maintaining a Healthy Server, and Appendices. The five parts are composed of thirteen chapters and three appendices.

The coverage of some network services spans multiple chapters. In particular, e–mail server coverage spans Chapter 1, Chapter 5, and Appendix C; and the topic of the Domain Name System

spans Chapter 4 and Appendix B. However, most topics are covered in a single chapter.

Although individual chapters can be read alone (for example, you could jump directly to Chapter 6 to read about the web server configuration file), the book was designed as a unit. Most chapters reference material covered in other chapters. When such a reference is made, it contains a pointer to the chapter that covers the referenced material. If you have a specific task to study, such as setting up a Samba server, feel free to jump directly to that topic. But, if like many system administrators, you need to support the entire range of Linux network services, you will benefit from reading the entire text.

Part 1: The Basics

All network services depend on the underlying operating system and the network hardware. In this part, we look at how the network hardware is configured, and the role that the startup process plays in initializing the hardware and starting the desired network services. Part 1 contains two chapters.

Chapter 1: The Boot Process

A description of the boot process is provided, including a description of Linux runlevels. This chapter describes the two most widely used Linux boot loaders (LILO and GRUB) and the lilo.conf and grub.conf files used to configure them. The role of the kernel in initializing hardware devices and the role of init in starting all of the system services are covered. init and the inittab configuration files are described, with emphasis on the key startup files that a network server administrator needs to understand.

Chapter 2: The Network Interface

An interface to the physical network is required for every network server. This chapter covers the installation and configuration of an Ethernet interface. Linux systems can also provide network support through the serial interface. The serial interface is described, along with the getty and login processes that support serial communications. TCP/IP can also be supported over serial line by PPP software. Both client and server PPP configurations are covered.

Part 2: Internet Server Configuration

Part 2 covers the configuration of the server side of traditional Internet services. The services covered in this part include Telnet, FTP, DNS, sendmail, Apache, gated, Zebra, and NAT. Part 2 is composed of five chapters.

Chapter 3: Login Services

Linux provides the complete range of traditional services that allow users to remotely log in to the server. Users with valid user accounts can log in remotely using telnet and ftp, if those services are running. Services such as telnet and ftp are started through inetd or xinetd. This chapter describes how users are given valid login accounts, and how inetd and xinetd are configured to start services on demand. Optional configuration for the WU–FTPD server is also touched on.

Chapter 4: Linux Name Services

The Domain Name System (DNS) is essential for the operation of your network. Linux provides the Berkeley Internet Name Domain (BIND) software that is the most widely used and most thoroughly

tested DNS server software available. This chapter provides detailed information on configuring the new BIND version 9 DNS software. It also covers the host table and how DNS and the host table are used together.

Chapter 5: Configuring a Mail Server

The most powerful and complex system for handling Internet mail service is sendmail. Most Linux distributions bundle sendmail as part of the system. This chapter shows you how to simplify a sendmail configuration by concentrating on what is important and how to create your own custom configuration.

Chapter 6: The Apache Web Server

The Apache web server, which is the most widely used web server in the world today, is included as part of the Linux distribution. This chapter explains the installation and configuration of a secure, reliable web service.

Chapter 7: Network Gateway Services

All internets require routers. Linux provides a full range of both static and dynamic routing. Various Linux distributions include the full–featured gateway daemon (gated) and the new Zebra suite of routing protocols. The configuration of both Zebra and gated are covered. Strengths and weaknesses of the RIP, RIPv2, OSPF, and BGP routing protocols offered by these packages are discussed. In addition to routing, the use of network address translation, which is available for Linux as "address masquerading," is described, and the way it is configured with iptables is covered.

Part 3: Departmental Server Configuration

Part 3 describes the configuration of services that are essential for a departmental server that supports desktop clients. DHCP, Samba, NFS, LPR/LPD, POP, IMAP, and procmail are covered in this part of the text. Part 3 contains four chapters.

Chapter 8: Desktop Configuration Servers

Configuring a TCP/IP client can be complex. A configuration server relieves your users of this task. Linux provides configuration servers for both Windows and Unix desktops through the Dynamic Host Configuration Protocol (DHCP) server. A Linux system can also act as a DHCP client. This chapter covers the configuration of both Linux client and server DHCP software.

Chapter 9: File Sharing

The most important feature of a departmental network is that it allows desktop computers to transparently share files. Linux provides this capability through the SAMBA server that provides native file sharing for Windows systems and through the NFS server that provides file sharing for Unix clients. This chapter provides detailed information about both of these services and about the Linux file system.

Chapter 10: Printer Services

Linux provides printer services to desktop clients through SAMBA and the Line Printer Daemon (LPD). Chapter 10 explains how printers are shared through these services, as well as how to install

and configure local printers.

Chapter 11: More Mail Services

Most desktop systems cannot directly receive Internet mail. They rely on a mailbox server to collect and hold the mail for them until they are ready to read it. Linux includes two techniques for providing this service. Post Office Protocol (POP), the traditional mailbox protocol, is still widely used. Internet Message Access Protocol (IMAP) has advanced features that make it very popular. Chapter 11 covers the installation, configuration, and administration of both services.

Part 4: Maintaining a Healthy Server

Part 4 focuses on tasks that are essential for maintaining a secure and reliable server, even if the tasks are not specifically linked to network services. Part 4 contains two chapters that cover security and troubleshooting.

Chapter 12: Security

A sad fact of life on the Internet is that there are people out there who will do you harm if they have the chance. To run a reliable server, you must run a secure server. This chapter tells you how to keep up−to−date on security issues, how to take advantage of the exceptionally good security features included in Linux, how to monitor your system for security problems, and how to add extra security features if you need them.

Chapter 13: Troubleshooting

Things can and will go wrong. When they do, you need to locate and fix the problem. Chapter 13 helps you test and debug the network, and analyze and resolve problems. It discusses when you need to upgrade your Linux kernel and how you can do it. It also describes the tools used to analyze network problems.

Part 5: Appendices

Part 5 concludes the book with a series of three appendices.

Appendix A: Installing Linux

This appendix provides information about installing Linux. Red Hat Linux is used as an example. This appendix is intended to provide installation information to those readers moving to Linux from Unix or Windows NT/2000.

Appendix B: BIND Reference

This appendix provides a summary of the BIND 9 configuration commands for the named.conf file. It also provides a summary of the BIND 8 configuration commands for administrators of Linux systems that are still running BIND 8. Understanding the differences between BIND 8 and BIND 9 syntax will also help administrators transitioning to the new software.

Appendix C: The *m4* Macros for *sendmail*

This appendix provides a summary of the m4 macros that are available to build a custom sendmail configuration.

Conventions

This book uses certain typographic styles to help you quickly identify important information and to avoid confusion over the meaning of words. This introduction shows an example of this in the use of a monospaced font when referring specifically to Linux commands. The following conventions are used throughout this book:

- A normal, proportionally spaced font is used for the bulk of the text in the book.
- *Italicized text* indicates technical terms that are introduced for the first time in a chapter. (Italics are also used for emphasis.)
- Monospaced text is used for listings and examples; and to identify the Linux commands, filenames, and domain names that occur within the body of the text.
- *Italicized monospaced text* is used in command syntax to indicate a variable for which you must provide the value. For example, a command syntax written as **HelpFile=***path* means that the variable name *path* must not be typed as shown; you must provide your own value for *path*.
- **Bold monospaced text** is used to indicate something that must be typed as shown. This might be user input in a listing, a recommended command line, or fixed values within the syntax of a command. For example, a command syntax written as **HelpFile=***path* means that the value **HelpFile=** must be typed exactly as shown.
- The square brackets in a command's syntax enclose an item that is optional. For example, **ls** [**–l**] means that –l is an optional part of the ls command.
- A vertical bar in a command's syntax means that you should chose one keyword or the other. For example, **true**|**false** means choose true or false.

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

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

Tip 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.

Warning 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 even restoring your entire system from scratch.

Sidebars

A Sidebar is like a Note, but is longer. Typically, a Note is one paragraph or less in length, but Sidebars are longer. The information in a Sidebar is useful, but doesn't fit into the main flow of the discussion.

Help Us Help You

Things change. In the world of computers, things change rapidly. Facts described in this book will become invalid over time. When they do, we need your help locating and correcting them. Additionally, a 600−page book is bound to have typographical errors. Let us know when you spot one. Send your improvements, fixes, and other corrections to support@sybex.com. To contact the author for information about upcoming books and talks on Linux, go to http://www.wrotethebook.com/.

Part I: The Basics

Chapter List

Chapter 1: The Boot Process
Chapter 2: The Network Interface

Part Overview

Featuring:

- The role that the ROM BIOS, MBR, and loader play in booting the system
- GRUB and LILO configuration
- How and why the kernel is passed parameters at boot time
- System runlevels and how they are configured by the inittab file
- The chkconfig and tksysv tools that control the startup scripts
- Loadable kernel modules and the tools that manage them
- How Ethernet device drivers are loaded and configured
- Configuring a network interface with ifconfig and the Red Hat Network Configuration tool
- How serial ports function and how they are used for networking
- PPP configuration and security
- Creating chat scripts

Chapter 1: The Boot Process

Overview

This chapter looks at what happens during a Linux boot. It examines the processes that take place and the configuration files that are read. Booting is a critical part of the operation of a server. The boot process brings all of the network hardware online and starts all of the network daemon processes when the system is powered–up. If the server will not boot, it is unavailable to all of the users and computers that depend on it. For this reason, it is essential that the administrator of a network server understand the boot process and the configuration files involved in that process. After all, you're the person who maintains those configuration files and who is responsible for recovering the system when it won't boot.

The term *boot* comes from *bootstrap loader*, which in turn comes from the old saying "pull yourself up by your bootstraps." The meaning of this expression is that you must accomplish everything on your own without any outside help. This is an apt term for a system that must start from nothing and finish running a full operating system. When the boot process starts, there is nothing in RAM—no program to load the system. The loader that begins the process resides in non–volatile memory. On PC systems, this means that the loader is part of the ROM BIOS.

Booting a Linux PC is a multistep procedure. It involves basic PC functions as well as Linux processes. This complex process begins in the PC ROM BIOS; it starts with the ROM BIOS program that loads the boot sector from the boot device. The boot sector either contains or loads a Linux boot loader, which then loads the Linux kernel. Finally, the kernel starts the init process, which loads all of the Linux services. The next few sections discuss this process in detail.

Note Two Linux loaders, LILO and GRUB, are covered in this chapter. LILO is given the bulk of the coverage because it is the default for most Linux distributions. GRUB is covered because it is the default loader for Red Hat Linux 7.2.

Loading the Boot Sector

The ROM BIOS is configured through the BIOS setup program. Setup programs vary among different BIOS versions, but all of them allow the administrator to define which devices are used to boot the system and the order in which those devices are checked. On some PC systems, the floppy drive and the first hard drive are the boot devices, and they are checked in that order. Systems that permit booting from the CD–ROM usually list the CD–ROM as the first boot device, followed by the first hard drive.

For an operational Linux server, set the ROM BIOS to check the floppy first and then the hard drive, even if you used a bootable CD–ROM for the initial installation. The reason for this is simple: The floppy is used to reboot an operational system when the hard drive is corrupted; the CD–ROM is only booted to install or upgrade the system software. During an installation, the system is offline, and you have plenty of time to fiddle with a BIOS setup program. But during an outage of an operational server, time is critical. You want to be able to reboot Linux and fix things as quickly as possible.

The first 512 bytes of a disk contain a boot sector. The ROM BIOS loads the boot sector from the boot device into memory, and transfers control to it. The bootstrap program from the boot sector then loads the operating system.

Floppy disks have only one boot sector, but hard disks may have more than one because each partition on a hard drive has its own boot sector. The first boot sector on the entire hard disk is called the *master boot record* (MBR). It is the only boot sector loaded from the hard drive by the ROM BIOS. The MBR contains a small loader program and a partition table. If the standard DOS MBR is used, it loads the boot sector from the active partition and then passes control to the boot sector. Thus, both the MBR and the active partition's boot sector are involved in the boot process.

Figure 1.1 shows how the boot process flows from the BIOS to the MBR and then to the partition's boot sector. This figure assumes a DOS MBR and a Linux loader in the boot sector of the active partition. Alternatively, the Linux loader can be installed in the MBR to eliminate one step in the boot process.

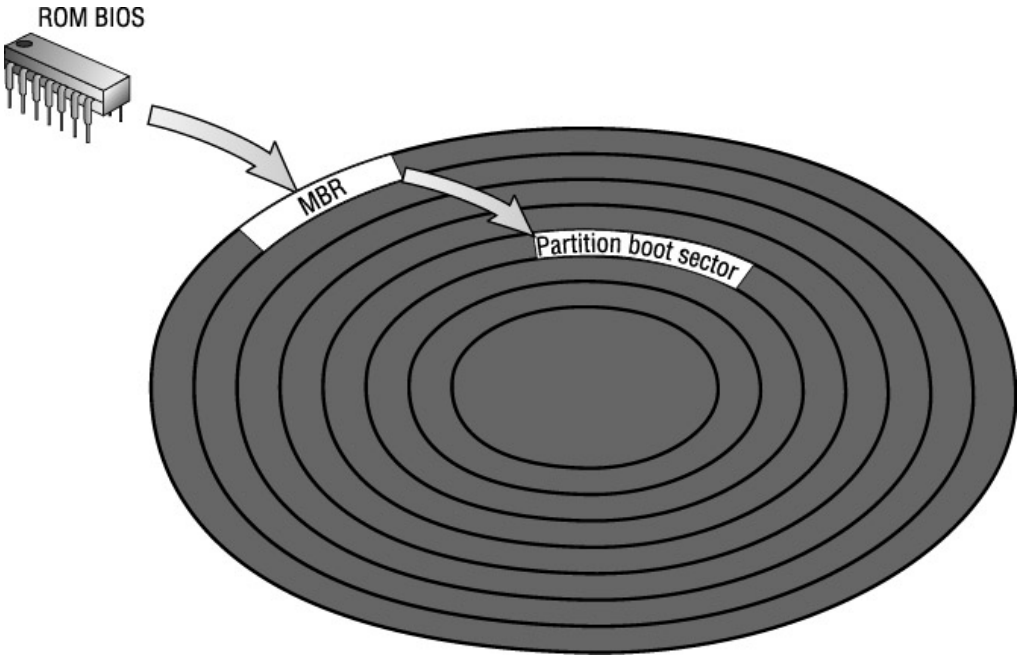


Figure 1.1: The boot process flow

Note Appendix A, "Installing Linux," discusses the pros and cons of placing the Linux loader in the MBR.

The BIOS may introduce some limitations into the Linux boot process. The Linux kernel can be installed anywhere on any of the disks available to the system, but if it is outside of those limits, the system might not be able to boot. The Linux loader depends on BIOS services. Some versions of BIOS only permit the loader to access the first two IDE hard drives: /dev/hda and /dev/ hdb. Additionally, in some cases, only the first 1024 cylinders of these disks can be used when booting the system. These limitations are at their worst on old systems. New systems have two IDE disk controllers that provide access to four disk drives, and these controllers address up to 8GB of disk storage within the 1024–cylinder limit. A very old system might address only 504MB in 1024 cylinders!

For a server installation, this is not a real problem. Because servers do not dual–boot, everything can be removed from the disk, and the Linux boot files can be installed in the first partition without difficulty.

A desktop client is a different matter. Most desktops have Microsoft Windows installed in the first partition. If there is available space within the first 1024 cylinders on the first disk drive, use fips to create empty space and install the Linux boot partition there. (Partitioning is discussed in detail in Appendix A.) Otherwise, a client system that dual–boots is forced to use one of the following methods:

- Install the Linux boot loader in the MBR of the first disk, and install the Linux boot partition in the first 1024 cylinders of the second disk.
- Use LOADLIN, SYSLINUX, System Commander, or a similar product to boot Linux from DOS instead of booting the system directly to Linux.
- Make a complete backup of Microsoft Windows, and repartition the disk so that both Windows and Linux are in the first 1024 cylinders. This, of course, requires a complete reinstallation of Windows.
- Create a Linux boot directory within the Windows directory structure that contains the Linux kernel and all of the files from the /boot directory.
- Upgrade the BIOS. This is not as difficult as it may sound. Most systems allow the BIOS to be upgraded, and many motherboard manufacturers and BIOS manufacturers have BIOS upgrades on their websites. However, don't undertake this lightly! A problem during the upgrade can leave the system unusable, and send you scurrying to the computer store to buy a replacement BIOS chip.
- Make a boot floppy or CD–ROM, and use that to start Linux. This is frequently the easiest option.

Don't be overly concerned about this potential problem. It is not a concern for servers, and even on clients it is rare. I have installed many Linux systems and have only had this problem once. In that case, it was a very old system that could directly address only 504MB per disk drive. My solution was to give the user a 250MB drive from my junk drawer as a second disk. (I never throw anything away.) I installed LILO in the MBR of his first disk and Linux on the second disk. The user was happy, Linux was installed, and I had less junk in my drawer.

Even though there are several options for loading Linux, only a few are widely used. Most systems use the Linux loader LILO. The Red Hat Linux 7.2 system defaults to using GRUB. This chapter covers both of these commonly used loaders. We start with a close look at the default GRUB configuration generated by the Red Hat installation program.

Loading Linux with GRUB

During the installation of Red Hat Linux 7.2, you're asked to select which boot loader should be used. By default, Red Hat uses the Grand Unified Bootloader (GRUB), and creates a GRUB configuration based on the values you select during the installation. Listing 1.1 shows the GRUB configuration generated by the Red Hat installation program for a desktop client. A dual–boot client configuration is used as an example because it is slightly more complex than a server configuration (servers do not usually dual–boot).

Listing 1.1: The Default GRUB Configuration

```
[root]# cat /etc/grub.conf
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE:  You do not have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /, eg.
#          root (hd0,2)
#          kernel /boot/vmlinuz-version ro root=/dev/hda3
#          initrd /boot/initrd-version.img
#boot=/dev/hda
default=0
timeout=10
splashimage=(hd0,2)/boot/grub/splash.xpm.gz
```

```
password --md5 $1$L°ÒCX≤Ëª$qgeIevUEDvvQAmrm4jCd31
title Red Hat Linux (2.4.7-10)
        root (hd0,2)
        kernel /boot/vmlinuz-2.4.7-10 ro root=/dev/hda3
        initrd /boot/initrd-2.4.7-10.img
title DOS
        rootnoverify (hd0,0)
        chainloader +1
```

The GRUB configuration is stored in grub.conf, which is a simple text file. Lines that begin with # are comments, and the Red Hat installation program inserts several comments at the beginning of the file.

The first active command line in this configuration is default=0. This command identifies which operating system should be booted by default in a dual–boot configuration. The operating systems that are available to GRUB are defined at the end of the configuration. Each operating system is assigned a number, sequentially starting from 0. Thus, the first operating system defined is 0, the second is 1, the third is 2, and so on. This configuration defines two operating systems: Red Hat Linux and DOS. Red Hat Linux is listed first; therefore, it is operating system 0, and it is the operating system that will be booted by default. In this case, the command default=0 is not really required because default is set to 0 whenever the default command is not included in the configuration. However, including the command makes a clean, self–documenting configuration.

The second active line, timeout=10, also relates to the default boot. The timeout command sets the number of seconds the operator has to interrupt the boot process before GRUB automatically loads the default operating system. In this example, the operator has 10 seconds to select the alternate operating system before Red Hat Linux is automatically booted. Even for systems that do not dual–boot, set a value for timeout because this allows the operator to interrupt the boot process if it is necessary to pass arguments to the kernel. Providing kernel input at the boot prompt is covered later in this chapter.

The splashimage command points to a file that contains the background image displayed by GRUB. During the timeout period, GRUB displays a boot menu. The splashimage file is the background displayed behind that menu.

During the initial installation of Red Hat Linux 7.2, you have an opportunity to enter a GRUB password. The password entered at that time is stored in the grub.conf file using the password command. The password "Wats?Watt?" was entered during the installation of our sample system. Note that the password is not stored as clear text. The password is encrypted, and the ––md5 option on the password command line lets us know that the password is encrypted with the Message Digest 5 (MD5) algorithm. The operator must enter the correct password to gain access to the full range of GRUB features. The operator can boot any of the operating systems listed in the GRUB menu without entering the password; however, optional input, such as kernel parameters, cannot be entered without the correct password. If the password command is not included in the grub.conf file, a password is not required to access any GRUB features.

The title command defines the exact text that will be displayed in the GRUB menu to identify an operating system. The commands that follow a title command and occur before the next title command describe an operating system to the boot loader. The sample configuration defines the following two operating systems:

```
title Red Hat Linux (2.4.7-10)
        root (hd0,2)
```

```
        kernel /boot/vmlinuz-2.4.7-10 ro root=/dev/hda3
        initrd /boot/initrd-2.4.7-10.img
title DOS
        rootnoverify (hd0,0)
        chainloader +1
```

The first title command defines the menu text Red Hat Linux (2.4.7–10). The next three lines define the operating system that is booted when that item is selected from the GRUB menu:

root (hd0,2) Defines the physical location of the filesystem root for this operating system. The values defined for the root command are the disk device name and the partition number. Notice that GRUB device names are slightly different from normal Linux device names. GRUB calls the first hard disk hd0. Additionally, GRUB counts partitions differently than Linux does. GRUB counts from 0, whereas Linux counts from 1. Thus, the GRUB value hd0,2 on a Linux system that boots from an IDE drive is the same as the Linux value hda,3—partition number 3 on the first IDE drive.

kernel /boot/vmlinuz–2.4.7–10 ro root=/dev/hda3 Identifies the file that contains the operating system that is to be started, and defines any arguments passed to that operating system at run time. In this case, GRUB will load the Linux kernel stored in vmlinuz–2.4.7–10, and it will pass the Linux kernel the arguments ro root=/dev/hda3, which tell the kernel where the filesystem root is located, and that it should be mounted as read–only. The ro option causes Linux to mount the root read–only during the initial phase of the boot. (Later, the rc.sysinit script changes it to read–write after successfully completing the filesystem checks.)

initrd /boot/initrd–2.4.7–10.img Identifies a ramdisk file for Linux to use during the boot. Red Hat uses the ramdisk to provide Linux with critical modules that the kernel might need to access the disk drives.

The last title command defines the DOS menu entry. Two commands define the operating system loaded when DOS is selected from the menu:

rootnoverify (hd0,0) Like the root command, defines the physical location of the filesystem root for this operating system. But rootnoverify tells GRUB that the filesystem found at this location does not comply with the multiboot standards, and thus cannot be validated.

chainloader +1 Emulates the function of the DOS MBR by simply loading the specified sector and passing boot responsibilities to the loader found there. The value +1 is a *blocklist* value, which defines the sector address of the loader relative to the partition defined by the rootnoverify command. +1 means the first sector of the partition. Taken together, the rootnoverify command and the chainloader command from our sample mean that GRUB will pass control to the loader found in the first sector of the first partition on the first IDE drive when DOS is selected from the GRUB menu. In this example, that partition contains the DOS boot loader that will be responsible for loading DOS.

The grub.conf file on your system will be very similar to the one in this example. The location of files may be different, and a server system's configuration usually won't define multiple operating systems, but the commands will be essentially the same.

GRUB is used with several different flavors of UNIX. It is not, however, the only boot loader used

with Linux—or even the most popular Linux boot loader. Red Hat, prior to 7.2, used LILO, and most other versions of Linux still do. The next section takes a close look at LILO configuration.

Loading the Kernel with LILO

Although GRUB is a newer tool, LILO, the Linux loader, is still a versatile tool that can manage multiple boot images; and can be installed on a floppy disk, in a hard disk partition, or as the master boot record. As with GRUB, this power and flexibility comes at the price of complexity, which is illustrated by the large number of LILO configuration options.

LILO Configuration Options

Most of the time, you don't need to think about the complexity of LILO; the installation program will lead you through a simple LILO installation. It is for those times when the default installation doesn't provide the service you want that you need to understand the intricacies of LILO.

LILO is configured by the /etc/lilo.conf file. Listing 1.2 is the lilo.conf file created by a Linux installation program on a desktop client that is configured to dual–boot. Its function is very similar to the GRUB sample shown in Listing 1.1.

Listing 1.2: A Sample *lilo.conf* File

```
# global section
boot=/dev/hda3
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
message=/boot/message
default=linux
# The Linux boot image
image=/boot/vmlinuz-2.4.7-10
    label=linux
    read-only
    root=/dev/hda3
# additional boot image
other=/dev/hda1
    optional
    label=dos
```

With this configuration, the user has five seconds to select either dos to boot Microsoft Windows or linux to boot Linux. If the user does not make a selection, LILO boots Linux after the five seconds have expired. The following section examines each line in this file to see how LILO is configured.

A Sample *lilo.conf* File

A lilo.conf file starts with a global section that contains options that apply to the entire LILO process. Some of these entries relate to the installation of LILO by /sbin/lilo, and are only indirectly related to the boot process.

Note The program /sbin/lilo is not the boot loader. The LILO boot loader is a simple loader stored in a boot sector. /sbin/lilo is the program that installs and updates the LILO boot

loader.

Comments in the lilo.conf file start with a sharp sign (#). The first active line of the global section in the sample file identifies the device that contains the boot sector. The option boot=/dev/hda3 says that LILO is stored in the boot sector of the third partition of the first IDE disk drive. This tells us two things: where LILO is installed and where it isn't installed. LILO is not installed in the MBR of this system; it is installed in hda3, which must be the active partition.

The configuration option map=/boot/map defines the location of the map file, which contains the physical locations of the operating system kernels in a form that can be read by the LILO boot loader. (GRUB does not require a map file because it can read Linux filesystems directly.) /boot/map is the default value for the map option, so, in this case, it does not really need to be explicitly defined in the sample configuration file.

The install=/boot/boot.b line defines the file that /sbin/lilo installs in the boot sector. (boot.b is the LILO boot loader.) In this case, the line is not actually required because /boot/boot.b is the default value for install.

The prompt option causes the boot prompt to be displayed. If the prompt option is not included in the lilo.conf file, the user must press a Shift, Ctrl, or Alt key; or set the Caps Lock or Scroll Lock key to get the boot prompt. The message displayed at the boot prompt is contained in the file identified by the message option. In the example, message points to a file named /boot/ message that contains a full−screen display. If the message option is not used, the default boot prompt boot: is used.

The timeout entry defines how long the system should wait for user input before booting the default operating system. The time is defined in tenths of seconds. Therefore, timeout=50 tells the system to wait five seconds.

Warning Don't use prompt without timeout. If the timeout option is not specified with the prompt option, the system will not automatically reboot. It will hang at the boot prompt, waiting for user input, and will never time out. This could be a big problem for an unattended server.

If the timeout is reached, the default kernel is booted. The default option identifies the default kernel. In Listing 1.2, the operating system that has the label "linux"is the one that will be started by default. To boot Microsoft Windows as the default operating system, simply change the default option to default=dos. The remainder of this configuration file provides the information that LILO needs to find and boot either Linux or Windows.

The image statement specifies the location of the Linux kernel, which is /boot/vmlinuz−2.4.7−10 in this example. The image option allows you to put the Linux kernel anywhere and name it anything. The ability to change the name of the kernel comes in very handy when you want to do a kernel upgrade, which is discussed in Chapter 13, "Troubleshooting."

There are several "per−image" options used in the configuration file, some of which are specific to kernel images. The label=linux option defines the label that is entered at the boot prompt to load this image. Every image defined in the sample file has an associated label entry; if the operator wants to boot an image, they must enter its label.

The next option, read−only, is also kernel−specific. It applies to the root filesystem described previously. The read−only option tells LILO that the root filesystem should be mounted read−only.

This protects the root filesystem during the boot and ensures that the filesystem check (fsck) runs reliably. Later in the startup process, the root will be re−mounted as read/write after fsck completes. See the discussion of rc.sysinit later in this chapter.

The root=/dev/hda3 option is also kernel−specific. It defines the location of the root filesystem for the kernel. The lilo.conf file should have a root option associated with the kernel image. If it is not defined here, the root filesystem must be defined separately with the rdev command. However, don't do that; define the root in the LILO configuration.

The last three lines in the sample file define the other operating system that LILO is able to boot. The other OS is located in partition 1 of the first IDE drive, other=/dev/hda1. As the label=dos entry indicates, it is Microsoft Windows. The optional command tells /sbin/lilo, which is called the *mapper*, that when it builds the map file, it should consider this operating system optional. That means that /sbin/lilo should complete building the map file, even if this operating system is not found.

Whenever you modify the LILO configuration, invoke /sbin/lilo to install the new configuration. Until /sbin/lilo is run and maps the new configuration options, they have no effect. The grub.conf file, on the other hand, does not require any special processing. Changes to the GRUB configuration take effect immediately.

Only Linux and one other operating system appear in the sample file, which is the most common case for desktop clients. However, LILO can act as the boot manager for up to 16 different operating systems. It is possible to see several other and image options in a lilo.conf file. Multiple image options are used when testing different Linux kernels. The most common reason for multiple other options is a training system in which users boot different OSs to learn about them. In an average operational environment, only one operating system is installed on a server, and no more than two operating systems are installed on a client.

lilo.conf Hardware Options

There are many more lilo.conf configuration options than those described previously, but you won't need to use most of them. The sample configuration file in Listing 1.2 is almost identical to the one built by the installation program on any other system. Basically, the small subset of options just described includes the options used to build 99 percent of all LILO configuration files.

The one percent of systems that cannot be configured with the usual commands are often those systems with hardware difficulties. The lilo.conf file provides several options for dealing with hardware problems.

The lba32 option is used when the boot partition is placed above the 1024−cylinder limit. This option requires a BIOS that supports 32−bit Logical Block Addresses (LBA32) for booting. The Red Hat installation program displays a "Force use of LBA32" check box in the boot loader installation screen. If this is available in your BIOS, it is the simplest way to boot from beyond the 1024−cylinder barrier.

The linear option forces the system to use linear sector addresses—sequential sector numbers—instead of traditional cylinder, head, and sector addresses. This is sometimes necessary to handle large SCSI disks. It is even possible to manually define the disk geometry and linear addresses of the partitions directly in the LILO configuration file. For example:

```
disk=/dev/hda
 bios=0x80
 sectors=63
```

```
 heads=32
 cylinders=827
 partition=/dev/hda1
  start=63
 partition=/dev/hda2
  start=153216
 partition=/dev/hda3
  start=219744
```

This example defines the geometry for the first disk drive, which normally has the BIOS address of hexadecimal 80. The sectors, heads, and cylinders of the disk are defined. In the example, the linear address for the start of each partition is also given. This is an extreme example of defining the disk drive for the system; I have never had to do this.

The append command is another LILO option related to defining hardware. (I have used this one.) The append option passes a configuration parameter to the kernel. The parameter is a kernel–specific option used to identify hardware that the system failed to automatically detect. For example:

```
append = "ether=10,0x210,eth0"
```

This sample command tells the kernel the nonstandard configuration of an Ethernet card. This particular option line says that the Ethernet device eth0 uses IRQ 10 and I/O port address 210. (The format of the parameters that can be passed to the kernel is covered in "The Linux Boot Prompt," later in this chapter.)

Linux is very good at detecting the configuration of Ethernet hardware, and software–configurable cards are good at reporting their settings. Additionally, new PCI cards do not require all of these configuration values. By and large, kernel parameters are not needed to boot the system. However, this capability exists for those times when you do need it.

LILO Boot Security

Two LILO configuration commands enhance the security of a network server. If the server is in an unsecured area, it is possible for an intruder to reboot the system and gain unauthorized access. For example, an intruder could reboot the server into single–user mode and essentially have password–free root access to part of the system. (More about single–user mode later. For now, just take my word that this can be done.)

To prevent this, add the password and the restricted options to the lilo.conf file. The password option defines a password that must be entered to reboot the system. The password is stored in the configuration file in an unencrypted format, so make sure the lilo.conf file can be read only by the root user. The restricted option softens the security a little. It says that the password is required only when passing parameters to the system during a boot. For example, if you attempt to pass the parameter single to the system to get it to boot into single–user mode, you must provide the password.

Always add the restrict option when using the password option in a server's lilo.conf file. Using password without restrict can cause the server to hang during the boot until the password is entered. If the server console is unattended, the boot can hang for an extended period of time. Using restrict with the password option ensures that the system reboots quickly after a crash, while providing adequate protection from unauthorized access through the console.

The following example includes restricted password protection for booting the Linux kernel. The example is based on the lilo.conf file you saw earlier, with a few lines removed that contain default values to show that you can remove those lines and still boot without a problem. Listing 1.3 uses cat to list the new configuration file and lilo to process it.

Listing 1.3: Adding Password Protection to LILO

```
[root]# cat lilo.conf
# global section
boot=/dev/hda3
prompt
timeout=50
message=/boot/message
default=linux
# the Linux boot image
image=/boot/vmlinuz-2.4.2-2
    label=linux
    read-only
    root=/dev/hda3
    password=Wats?Watt?
    restricted
# additional boot images
other=/dev/hda1
    optional
    label=dos
 [root]# lilo
Added linux *
Added dos
```

After running /sbin/lilo, reboot. Note that you don't have to enter the password at the boot prompt because the configuration includes the restrict option. However, if you attempt to boot the system and provide optional input at the boot prompt, you will be asked for the password.

The Linux Boot Prompt

The LILO and GRUB processes are modified through their configuration files. The kernel boot process is modified through input to the boot prompt. As with the LILO append option and the GRUB kernel command, the boot prompt is used to pass parameters to the kernel. The difference, however, is that the boot prompt is used to manually enter kernel parameters, whereas the append and kernel commands are used to automate the process when the same parameters must be passed to the kernel for every boot. Use the boot prompt for special situations, such as repairing a system or getting an unruly piece of equipment running; or to debug input before it is stored in the lilo.conf or grub.conf file.

You rarely need to pass parameters to the kernel through the boot prompt. When you do, it is either to change the boot process or to help the system handle a piece of unknown hardware. The kernel command from the grub.conf file shown in Listing 1.1 is an example of using boot input to change the boot process:

```
kernel /boot/vmlinuz-2.4.7-10 ro root=/dev/hda3
```

This line comes from the grub.conf file, but it also can be entered interactively during the boot process. When the GRUB menu is displayed at boot time, the operator is given 10 seconds to

select an optional menu item, or interrupt the boot process. Interrupt the boot by pressing the Escape key. If a password is defined in the grub.conf file, press P, and enter the GRUB password. Then, press C for command mode, and a command line prompt appears. This is the boot prompt that allows arguments to be sent to the kernel using the kernel command interactively. The format of the kernel command is

kernel *file arguments*

where kernel is the command, *file* is the name of the file that contains the Linux kernel, and *arguments* are any optional arguments you wish to pass to the kernel. In the preceding kernel command example, ro root=/dev/hda3 are arguments that change the default boot behavior so that the root filesystem is mounted read−only. The possible arguments depend on the kernel, not on whether GRUB or LILO is used to control the boot process. Any of the kernel arguments described in this section can be sent to the kernel in this manner on a system that uses GRUB. The LILO boot prompt is different, but the function is the same.

When the system is booted by LILO, the string boot: is displayed as the boot prompt. The operator can boot any operating system defined in the lilo.conf file by entering its name at the prompt (for example, linux, or dos). Arguments are passed to the selected operating system by placing them on the command line after the operating system name. An example of passing kernel parameters on a system booted by LILO is

```
boot: linux panic=60
```

In this example, boot: is the prompt, linux is the kernel name, and panic=60 is the parameter passed to that kernel. The keyword linux is the label assigned to the Linux kernel in the LILO configuration. Use the label to tell LILO which kernel should receive the parameter. The panic argument changes the boot behavior after a system crash. It is possible for the Linux kernel to crash from an internal error, called a *kernel panic*. If the system crashes from a kernel panic, it does not automatically reboot—it stops at the boot prompt waiting for instructions.

Normally, this is a good idea. The exception is an unattended server. If you have a system that does not have an operator in attendance and that remote users rely on, it might be better to have it try an automatic reboot after it crashes. The example shown previously tells the system to wait 60 seconds and then reboot.

Note This might surprise Windows administrators, but I have never had a Linux system crash. In fact, I had one specialized system (collecting network measurement data, and providing Web access to that data) that ran continuously for more than a year without a single problem.

In a normal boot process, the kernel starts the /sbin/init program. Using the init argument, it is possible to tell the kernel to start another process instead of /sbin/init. For example, init=/bin/sh causes the system to run the shell program, which then can be used to repair the system if the /sbin/init program is corrupted.

Booting directly to the shell looks very much like booting to single−user mode with the single argument, but there are differences. init=/bin/sh does not rely on the init program. single, on the other hand, is passed directly to init so that init can perform selected initialization procedures before placing the system into single−user mode. In both of these cases, the person who boots the computer is given password−free access to the shell unless password and restrict are defined in the lilo.conf file, as described in the previous section.

Handling undetected hardware is the second reason for entering data at the boot prompt, and it is the most common reason for doing so during the initial installation. Sometimes, the system has trouble detecting hardware or properly detecting the hardware's configuration. In those cases, the system needs your input at the boot prompt to properly handle the unknown hardware.

A large number of the boot input statements pass parameters to device driver modules. For example, there are about 20 different SCSI host adapter device drivers that accept boot parameters. In most cases, the system detects the SCSI adapter configuration without a problem. But if it doesn't, booting the system may be impossible. An example of passing kernel parameters to Linux to identify an undetected SCSI adapter device is

```
boot: linux aha152x=0x340,11,7
```

All hardware parameters begin with a driver name. In this case, it is the aha152x driver for Adaptec 1520 series adapters. The data after the equal sign is the information passed to the driver. In this case, it is the I/O port address, the IRQ, and the SCSI ID.

Another boot argument that is directly related to the configuration of device drivers is the reserve argument. reserve defines an area of I/O port address memory that is protected from *auto–probing*. To determine the configuration of their devices, most device drivers probe those regions of memory that can be legitimately used for their devices. For example, the 3COM EtherLink III Ethernet card is configured to use I/O port address 0x300 by default, but it can be configured to use any of 21 different address settings from 0x200 to 0x3e0. If the 3c509 driver did not find the adapter installed at address 0x300, it could legitimately search all 21 base address regions. Normally, this is not a problem. On occasion, however, auto–probing can return the wrong configuration values. In extreme cases, poorly designed adapters can even hang the system when they are probed. I have never personally seen an adapter hang the system, but some years ago I had an Ethernet card that returned the wrong configuration. In that case, I combined the reserve argument with device driver input, as in this example:

```
boot: linux reserve=0x210,16 ether=10,0x210,eth0
```

This boot input prevents device drivers from probing the 16 bytes starting at memory address 0x210. The second argument on this line passes parameters to the ether device driver. It tells that driver that the Ethernet adapter uses interrupt 10 and I/O port address 0x210. This specific adapter will be known as device eth0, which is the name of the first Ethernet device. Of course, you'll want to use the Ethernet adapter every time the system boots. Once you're sure this boot input fixes the Ethernet problem, store it as a kernel–specific option in the lilo.conf file. For example:

```
image = /boot/vmlinuz-2.2.5-15
    label = linux
    root = /dev/hda3
    read-only
    append = "reserve=0x210,16 ether=10,0x210,eth0"
```

The ether argument is also used to force the system to locate additional Ethernet adapters. Suppose that the system detects only one Ethernet adapter, and you have two Ethernet devices installed: eth0 and eth1. Use this boot input to force the system to probe for the second device:

```
ether=0,0,eth1
```

Old Ethernet cards are a major reason for boot prompt input. If you have an old card and experience a problem, read the Ethernet–HOWTO for configuration advice on your specific card. New PCI Ethernet cards do not usually require boot input. Most current Ethernet cards use loadable

modules for device drivers. If your Ethernet card is not recognized during the boot, it may be that its module is not loaded. The first step is to check the module's configuration.

Note See the "Loadable Modules" section later in this chapter for information about managing modules and for specific examples of loadable modules used for Ethernet device drivers.

This section has barely touched upon the very large number of arguments that can be entered at the boot prompt. See the "BootPrompt–HOWTO" document, by Paul Grotmaker, for the details of all of them. Most Linux systems include the HOWTO documents in /usr/doc.

Hardware Device Driver Initialization

When the system boots, several things happen. You have already seen the part that LILO and GRUB play in loading the operating system, but that is only the beginning. These loaders start the Linux kernel running, and then things *really* begin to happen.

The kernel is the heart of Linux. It loads into memory and initializes the various hardware device drivers. Most of the possible boot prompt arguments are intended to help the kernel initialize hardware, and the messages the kernel displays during startup help you determine what hardware is installed in the system and whether it is properly initialized.

Use the dmesg command to display the kernel startup messages; combine it with the less command or with grep to examine the startup messages more effectively. less allows you to scroll through the messages, one screenful at a time; grep permits you to search for something specific in the dmesg output. For example, combine dmesg and grep to locate kernel messages relating to the initialization of the Ethernet device eth0:

```
$ dmesg | grep eth0
loading device 'eth0'...
eth0: SMC Ultra at 0x340, 00 00 C0 4F 3E DD, IRQ 10 memory 0xc8000-0xcbfff.
```

This message clearly shows the type of Ethernet adapter used (SMC Ultra) and the Ethernet MAC address assigned to the adapter (00 00 C0 4F 3E DD). Additionally, because the SMC Ultra is an ISA bus adapter, the bus interrupt (IRQ 10), the adapter memory address (0xc8000–0xcbfff), and the I/O port address (0x340) are shown. All of this information is useful for debugging a hardware configuration problem.

Most systems use Ethernet for all network communications, although some use other devices, such as serial ports for this purpose. When the kernel initializes the serial ports, it displays the device name, I/O port address, and IRQ of each serial port. It also displays the model of Universal Asynchronous Receiver Transmitter (UART) that is used for the serial interface. Old systems used 8250 UARTs, which are inadequate for use with modems and a problem for systems that need to run PPP. As this example shows, current systems use the faster 16550A UARTs:

```
ttyS00 at 0x03f8 (irq = 4) is a 16550A
ttyS02 at 0x03e8 (irq = 4) is a 16550A
```

Also of interest for a network server are the kernel components of TCP/IP. These components include the fundamental protocols, such as IP (Internet Protocol), and the network sockets interface. Sockets is an application protocol interface developed at Berkeley for BSD Unix. It provides a standard method for programs to talk to the network. The TCP/IP initialization messages from a Red Hat 7.2 system are

```
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP, IGMP
IP: routing cache hash table of 1024 buckets, 8Kbytes
TCP: Hash tables configured (established 16384 bind 16384)
Linux IP multicast router 0.06 plus PIM-SM
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
```

Reading the kernel messages helps you understand what occurs when the system starts up. Don't read these messages word for word—too many details will just bog you down. What you should do is look at the messages to gain a sense of how the system works. Of course, there are slight variations among the messages displayed on various systems, but the messages give you a very good idea of what is going on as the kernel initializes the hardware.

After the kernel concludes its portion of the boot process, the kernel starts the init program, which controls the rest of the startup.

Loading Linux Services—The *init* Process

The init process, which is process number one, is the mother of all processes. After the kernel initializes all of the devices, the init program runs and starts all of the software. The init program is configured by the /etc/inittab file. Listing 1.4 shows the inittab file that comes with Red Hat 7.2:

Listing 1.4: The *inittab* File

```
#
# inittab   This file describes how the INIT process should set up
#           the system in a certain run-level.
#
# Author:   Miquel van Smoorenburg, <miquels@drinkel.nl.mugnet.org>
#           Modified for RHS Linux by Marc Ewing and Donnie Barnes
#

# Default runlevel. The runlevels used by RHS are:
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser, without NFS (The same as 3, if you do not have networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
#
id:5:initdefault:

# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit

l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6

# Things to run in every runlevel.
ud::once:/sbin/update

# Trap CTRL-ALT-DELETE
```

```
ca::ctrlaltdel:/sbin/shutdown -t3 -r now

# When our UPS tells us power has failed, schedule a shutdown for 2 minutes.
pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting Down"

# If power was restored before the shutdown, cancel it.
pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown Cancelled"

# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6

# Run xdm in runlevel 5
# xdm is now a separate service
x:5:respawn:/etc/X11/prefdm -nodaemon
```

Note The comments in this sample file were edited slightly to better fit on a book page. They are a reduced version of the actual comments from the Red Hat inittab file.

Understanding Runlevels

To understand the init process and the inittab file, you need to understand *runlevels*, which are used to indicate the state of the system when the init process is complete. There is nothing inherent in the system hardware that recognizes runlevels; they are purely a software construct. init and inittab are the only reasons why the runlevels affect the state of the system. Because of this, the way runlevels are used varies from distribution to distribution. This section uses Red Hat Linux as an example.

The Linux startup process is very similar to the startup process used by System V Unix. It is more complex than the initialization on a BSD Unix system, but it is also more flexible. Like System V, Linux defines several runlevels that run the full gamut of possible system states from not–running (halted) to running multiple processes for multiple users. The comments at the beginning of the sample inittab file describe the runlevels:

- Runlevel 0 causes init to shut down all running processes and halt the system.
- Runlevel 1 is used to put the system in single–user mode. Single–user mode is used by the system administrator to perform maintenance that cannot be done when users are logged in. This runlevel may also be indicated by the letter S instead of the number 1.
- Runlevel 2 is a special multiuser mode that supports multiple users but does not support file sharing.
- Runlevel 3 is used to provide full multiuser support with the full range of services. It is the default mode used on servers that use the "text only" console logon.
- Runlevel 4 is unused by the system. You can design your own system state and implement it through runlevel 4.
- Runlevel 5 initializes the system as a dedicated X Windows terminal. This runlevel is widely used as an alternative for systems configured to launch an X desktop environment at startup. In fact, runlevel 5 is the default runlevel for most Red Hat systems because most systems are desktop clients that use an X Windows console logon.
- Runlevel 6 causes init to shut down all running processes and reboot the system.

All of the lines in the inittab file that begin with a sharp sign (#) are comments. A liberal dose of comments is needed to interpret the file because the syntax of actual inittab configuration lines is

terse and somewhat arcane. An inittab entry has this general format:

```
label:runlevel:action:process
```

The *label* is a one− to four−character tag that identifies the entry. Some systems support only two−character labels. For this reason, most people limit all labels to two characters. The labels can be any arbitrary character string, but in practice, certain labels are commonly used. The label for a getty or other login process is usually the numeric suffix of the tty to which the process is attached. Other labels used in the Red Hat Linux distribution are

- id for the line that defines the default runlevel used by init
- si for the system initialization process
- l*n* where *n* is a number from 1 to 6 that indicates the runlevel being initialized by this process
- ud for the update process
- ca for the process run when Ctrl+Alt+Del is pressed
- pf for the process run when the UPS indicates a power failure
- pr for the process run when power is restored by the UPS before the system is fully shut down
- x for the process that turns the system into an X terminal

The *runlevel* field indicates the runlevels to which the entry applies. For example, if the field contains a 3, the process identified by the entry must be run for the system to initialize runlevel 3. More than one runlevel can be specified, as illustrated in the sample file by the pr entry. Entries that have an empty *runlevel* field are not involved in initializing specific runlevels. For example, an entry that is invoked by a special event, such as the three−finger salute (Ctrl+Alt+Del), does not have a value in the *runlevel* field.

The *action* field defines the conditions under which the process is run. Table 1.1 lists all of the valid action values and the meaning of each one.

Table 1.1: Valid Action Values

Action	Meaning
Boot	Runs when the system boots. Ignores runlevel.
Bootwait	Runs when the system boots, and init waits for the process to complete. Runlevels are ignored.
Ctrlaltdel	Runs when Ctrl+Alt+Del is pressed, which passes the SIGINT signal to init. Runlevels are ignored.
Initdefault	Doesn't execute a process. It sets the default runlevel.
Kbrequest	Runs when init receives a signal from the keyboard. This requires that a key combination be mapped to KeyBoardSignal.
Off	Disables the entry so the process is not run.
Once	Runs one time for every runlevel.
Ondemand	Runs when the system enters one of the special runlevels A, B, or C.
Powerfail	Runs when init receives the SIGPWR signal.
Powerokwait	Runs when init receives the SIGPWR signal and the file /etc/ powerstatus contains the word OK.
Powerwait	Runs when init receives the SIGPWR signal, and init waits for the process to complete.
Respawn	Restarts the process whenever it terminates.

sysinit	Runs before any boot or bootwait processes.
wait	Runs the process upon entering the run mode, and init waits for the process to complete.

The last field in an inittab entry is the process field. It contains the process that init executes. The process appears in the exact format that is used to execute the process from the command line. Therefore, the process field starts with the name of the process that is to be executed, and follows it with the arguments that will be passed to that process. For example, /sbin/shutdown –t3 –r now, which is the process executed when Ctrl+Alt+Del is pressed, is the same command that could be typed at the shell prompt to reboot the system.

Special–Purpose Entries

Using what you have just learned about the syntax of the inittab file, take a closer look at the sample in Listing 1.4. You can ignore most of the file; more than half of it consists of comments. Many of the other lines are entries that are used only for special functions:

- The id entry defines the default runlevel, which is usually 3 for a text console or 5 for an X console.
- The ud entry calls the /sbin/update process, which cleans up the I/O buffers before disk I/ O starts in order to protect the integrity of the disks.
- The pf, pr, and ca entries are invoked only by special interrupts.

Warning Some administrators are tempted to change the ca entry to eliminate the ability to reboot the system with the three–finger salute. This is not a bad idea for server systems, but don't do it for desktop systems. Users need to have a method to force a graceful shutdown when things go wrong. If it is disabled, the user might resort to the power switch, which can result in lost data and other disk troubles.

Six of the lines in the inittab file start—and when necessary, restart—the getty processes that provide virtual terminal services. One example from Listing 1.4 explains them all:

```
3:2345:respawn:/sbin/mingetty tty3
```

The label field contains a 3, which is the numeric suffix of the device, tty3, to which the process is attached. This getty is started for runlevels 2, 3, 4, and 5. When the process terminates (for example, when a user terminates the connection to the device), the process is immediately restarted by init.

The pathname of the process that is to be started is /sbin/mingetty. Red Hat uses mingetty, which is a minimal version of getty that is specifically designed for virtual terminal support. On a Caldera 2.2 system, the pathname would be /sbin/getty with the VC command–line option, which tells getty that it is servicing a virtual terminal. The result, however, would be the same: to start a virtual terminal service process for tty3.

Every runlevel that accepts terminal input uses getty. Runlevel 5 has one additional entry in the inittab file to start an X terminal:

```
x:5:respawn:/etc/X11/prefdm -nodaemon
```

This line starts—and when necessary, restarts—the X application used for the X–based console logon required by runlevel 5.

Every line in the inittab file handles some important task. However, the real heart of the inittab file consists of the seven lines that follow the comment "System initialization" near the beginning of the inittab file (refer to Listing 1.4.) They are the lines that invoke the startup scripts. The first of these is the si entry:

```
si::sysinit:/etc/rc.d/rc.sysinit
```

This entry tells init to initialize the system by running the boot script located at /etc/rc.d/rc.sysinit. This script, like all startup scripts, is an executable file that contains Linux shell commands. Notice that the entry shows the full path to the startup script. One of the most common complaints about different Linux distributions is that the key files are stored in different locations in the filesystem. Don't worry about memorizing these differences—just look in the /etc/inittab file. It tells you exactly where the startup scripts are located.

The six lines that follow the si entry in inittab are used to invoke the startup scripts for each runlevel. Except for the runlevel involved, each line is identical:

```
l5:5:wait:/etc/rc.d/rc 5
```

This line starts all of the processes and services needed to provide the full multiuser support defined by runlevel 5. The label is l5, which is symbolic of level 5. The runlevel is, of course, 5. init is directed to wait until the startup script terminates before going on to any other entries in the inittab file that relate to runlevel 5. init executes the script /etc/rc.d/rc, and passes that script the command–line argument 5.

Startup Scripts

Anything that can be run from a shell prompt can be stored in a file and run as a shell script. System administrators use this capability to automate all kinds of processes; Linux uses this capability to automate the startup of system services. Two main types of scripts are used: the *system initialization* script and the *runlevel initialization* script.

System Initialization

The system initialization script runs first. On a Red Hat system, this is a single script named /etc/rc.d/rc.sysinit. Other Linux distributions might use a different filename, but all versions of Linux use script files to initialize the system. The rc.sysinit script performs many essential system initialization tasks, such as preparing the network and the filesystems for use.

The rc.sysinit script begins the network initialization by reading the /etc/sysconfig/network file, which contains several network configuration values set during the initial installation. If the file is not found, networking is disabled. If it is found, the script assigns the system the hostname stored there.

The initialization script performs many small but important tasks, such as setting the system clock, applying any keyboard maps, and starting USB and PnP support. The bulk of the script, however, is used to prepare the filesystem for use. The script activates the swap file, which is necessary before the swap space is used. The rc.sysinit script also runs the filesystem check, using the fsck command to check the structure and integrity of the Linux filesystems. If a filesystem error is

encountered that fsck cannot simply repair, the boot process stops, and the system reboots in single−user mode. You then must run fsck manually, and repair the disk problems yourself. When you finish the repairs, exit the single−user shell. The system will then attempt to restart the interrupted boot process from where it left off.

The initialization script mounts the /proc filesystem and, after the fsck completes, mounts the root filesystem as read−write. Recall that the root filesystem was initially mounted as read−only. The root must be remounted as read−write before the system can be used. The script also mounts other local filesystems listed in the /etc/fstab file. (The fstab file is described in Chapter 9, "File Sharing.") The rc.sysinit script finishes up by loading the loadable kernel modules.

Other initialization scripts may look different from Red Hat's, but they perform very similar functions. The order may be different, but the major functions are the same: initialize the swap file, and check and mount the local filesystems.

Runlevel Initialization

After the system initialization script is run, init runs a script for the specific runlevel. On Red Hat, Mandrake, and Caldera systems, this is done by running a control script and passing it the runlevel number. The control script, /etc/rc.d/rc, then runs all of the scripts that are appropriate for the runlevel. It does this by running the scripts that are stored in the directory /etc/rc*n.d*, where *n* is the specified runlevel. For example, if the rc script is passed a 5, it runs the scripts found in the directory /etc/rc.d/rc5.d. A listing of that directory from a Red Hat system shows that there are lots of scripts:

Listing 1.5: Runlevel Initialization Scripts

```
$ ls /etc/rc.d
init.d  rc0.d  rc2.d  rc4.d  rc6.d     rc.sysinit
rc      rc1.d  rc3.d  rc5.d  rc.local
$ ls /etc/rc.d/rc3.d
K03rhnsd      K35smb       K74ntpd    S05kudzu     S25netfs       S85httpd
K16rarpd      K45arpwatch  K74ypserv  S06reconfig  S26apmd        S90crond
K20nfs        K45named     K74ypxfrd  S08ipchains  S28autofs      S90xfs
K20rstatd     K50snmpd     K75gated   S09isdn      S40atd         S95anacron
K20rusersd    K50tux       K84bgpd    S10network   S55sshd        S99linuxconf
K20rwalld     K55routed    K84ospf6d  S12syslog    S56rawdevices  S99local
K20rwhod      K61ldap      K84ospfd   S13portmap   S56xinetd
K28amd        K65identd    K84ripd    S14nfslock   S60lpd
K34yppasswdd  K73ypbind    K84ripngd  S17keytable  S80sendmail
K35dhcpd      K74nscd      K85zebra   S20random    S85gpm
```

The scripts that begin with a K are used to kill processes when exiting a specific runlevel. In Listing 1.5, the K scripts are used when terminating runlevel 5. The scripts that start with an S are used when starting runlevel 5. None of the items in rc5.d, however, is really a startup script. They are logical links to the real scripts, which are located in the /etc/rc.d/init.d directory. For example, S80sendmail is linked to /etc/init.d/sendmail. This raises the question of why the scripts are executed from the directory rc5.d instead of directly from init.d, where they actual reside. The reasons are simple. The same scripts are needed for several different runlevels. Using logical links, the scripts can be stored in one place and still be accessed by every runlevel from the directory used by that runlevel. Additionally, the order in which the scripts are executed is controlled by the script name.

The scripts are executed in alphabetical order, based on name. Thus, S10network is executed

before S80sendmail. This allows the system to control, through a simple naming convention, the order in which scripts are executed. Different runlevels can execute the scripts in different orders while still allowing the real scripts in init.d to have simple, descriptive names. Listing 1.6 shows the real script names in the init.d directory:

Listing 1.6: The init.d Script Files

```
$ ls init.d
amd       functions  kdcrotate  network  rarpd       rwalld    xfs
anacron   gated      keytable   nfs      rawdevices  rwhod     xinetd
apmd      gpm        killall    nfslock  reconfig    sendmail  ypbind
arpwatch  halt       kudzu      nscd     rhnsd       single    yppasswdd
atd       httpd      ldap       ntpd     ripd        smb       ypserv
autofs    identd     linuxconf  ospf6d   ripngd      snmpd     ypxfrd
bgpd      ipchains   lpd        ospfd    routed      sshd      zebra
crond     iptables   named      portmap  rstatd      syslog
dhcpd     isdn       netfs      random   rusersd     tux
```

Several of these scripts are clearly of interest to administrators of network servers:

- The httpd script starts the Web server.
- The xinet script starts the Extended Internet daemon (xinetd).
- The named script starts the DNS name server.
- The nfs script starts the NFS file server.
- sendmail starts the e–mail server.

It is useful to know where these services really start in case something goes wrong. All of these scripts may be important when troubleshooting a network problem.

Controlling Scripts

You can control which scripts are executed and the order in which they are executed by directly changing the logical links in the runlevel directory, but that's not the best way. It's easier to control startup scripts using a tool specifically designed for this purpose. Red Hat systems use the chkconfig command, which is a command–line tool based on the chkconfig program from the Silicon Graphics IRIX version of Unix. The Linux version has some enhancements, such as the capability to control which runlevels the scripts run under. The ––list option of the chkconfig command displays the current settings:

```
 [root]# chkconfig --list named
named 0:off 1:off 2:off 3:on 4:on 5:on 6:off
```

This example shows the structure of a chkconfig command line. chkconfig is the command, ––list is the option, and named is the name of a script file found in the init.d directory. It is the script file that the command affects.

To enable or disable a script for a specific runlevel, specify the runlevel with the ––level option, followed by the name of the script you wish to control and the action you wish to take, either on to enable the script or off to disable it. For example, to disable named for runlevel 5, enter the following:

```
[root]# chkconfig --level 5 named off
[root]# chkconfig --list named
named 0:off 1:off 2:off 3:on 4:on 5:off 6:off
```

chkconfig reads the comments in the init.d script file to determine the runlevels in which the script is run by default, and to obtain information needed to create the correct logical links in the runlevel directory. This information must be found in the script file in a comment that contains the keyword chkconfig. Here is an example from the ipchains script:

```
[root]# grep chkconfig ipchains
# chkconfig: 2345 08 92
```

In this comment, the keyword chkconfig is followed by three values:

- First, the list of runlevels in which this script is run by default. Here, the list contains four runlevels (2, 3, 4, and 5). If the script is not run by default at any runlevel, this field contains a dash (–).
- Next, the numeric prefix used to name the logical link to the script file used during startup. Here, the numeric prefix used for startup is 08. Therefore, the link placed in the runlevel directory will be named S08ipchains.
- Finally, the numeric prefix used to name the logical link to the script file used during shutdown. Here, the numeric prefix used for shutdown is 92. Therefore, the link placed in the runlevel directory will be named K92ipchains.

Editing the chkconfig comment in the script in the init.d directory changes the values that chkconfig uses to create the links. However, this is not necessary. The values selected by Red Hat were chosen to ensure that services start in the proper order. The only time you may need to set these values is when you write your own startup script for a custom service.

chkconfig is used on Red Hat and several other Linux systems. It is not, however, the only widely used tool for controlling scripts. tksysv, the SYSV Runlevel Manager, is available on several distributions; and it runs under X Windows. Figure 1.2 shows the SYSV Runlevel Manager window.

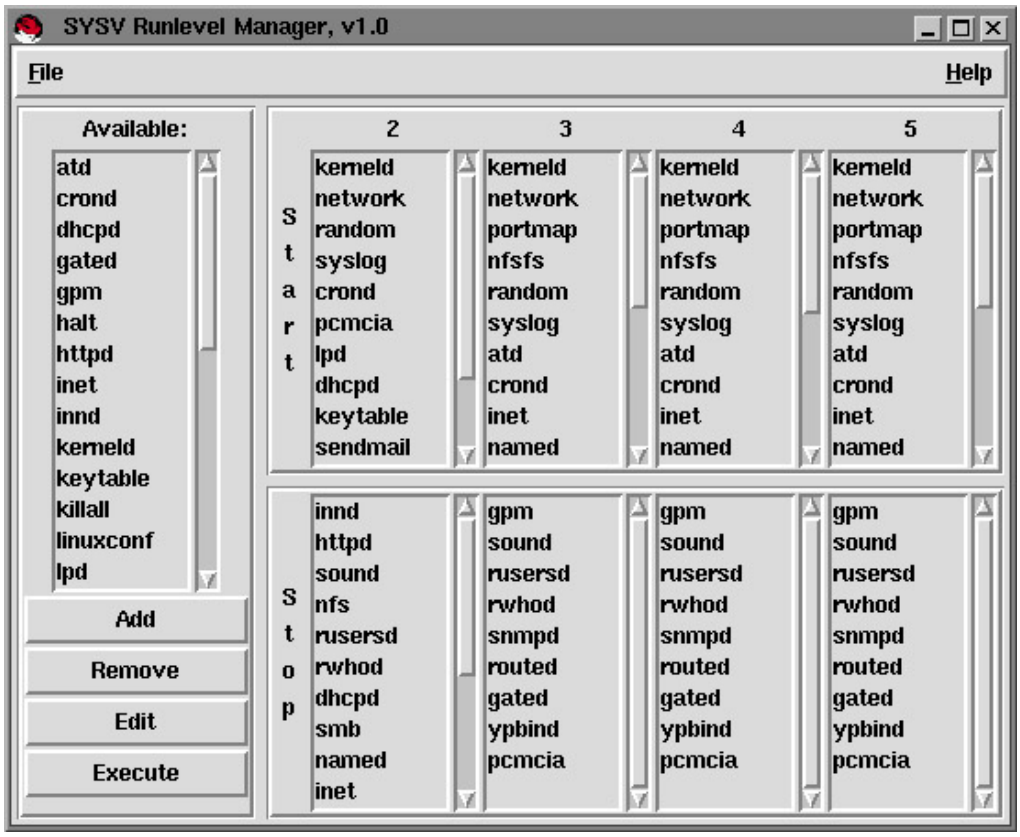


Figure 1.2: The SYSV Runlevel Manager Window
The SYSV Runlevel Manager lists all of the available startup scripts, as well as the scripts that are currently being used by each runlevel. Each runlevel has a column of the display that is divided into Start and Stop scripts. These categories correspond to the S and K scripts in the directories. Using tksysv's simple visual interface, you can add scripts to a runlevel from the list of available scripts, or delete scripts from a runlevel. You can even select a script from the list of available scripts and execute it in real time to start a service without rebooting.

The *rc.local* Script

In general, you do not directly edit boot scripts. The exception to this rule is the rc.local script located in the /etc/rc.d directory. It is the one customizable startup file, and it is reserved for your use; you can put anything you want in there. After the system initialization script and the runlevel scripts execute, the system executes rc.local. Since it is executed last, the values you set in the rc.local script are not overridden by another script.

If you add third–party software that needs to be started at boot time, put the code to start it in the rc.local script. Additionally, if something is not installed or configured correctly by the installation process, it can be manually configured in rc.local.

Loadable Modules

Loadable modules are pieces of object code that can be loaded into a running kernel. This is a very powerful feature. It allows Linux to add device drivers to a running Linux system in real time. This means that the system can boot a generic Linux kernel and then add the drivers needed for the hardware on a specific system. The hardware is immediately available without rebooting the system.

Usually, you have very little involvement with loadable modules. In general, the system detects your hardware and determines the correct modules during the initial installation. But not always. Sometimes hardware is not detected during the installation, and other times new hardware is added to a running system. To handle these things, you need to know how to work with loadable modules.

Listing the Loaded Modules

Use the lsmod command to check which modules are loaded in your system. Listing 1.7 shows an example:

Listing 1.7: Listing Loaded Modules

```
$ lsmod
Module                  Size  Used by
ide-cd                 27072   0  (autoclean)
cdrom                  28512   0  (autoclean) [ide-cd]
soundcore               4464   0  (autoclean)
parport_pc             14768   1  (autoclean)
lp                      6416   0  (autoclean)
parport                25600   1  (autoclean) [parport_pc lp]
autofs                 11520   0  (autoclean) (unused)
smc-ultra               5792   1
8390                    6752   0  [smc-ultra]
nls_iso8859-1           2832   1  (autoclean)
nls_cp437               4352   1  (autoclean)
```

```
vfat                 9584   1  (autoclean)
fat                 32384   0  (autoclean) [vfat]
ext3                64624   3
jbd                 40992   3  [ext3]
```

Loadable modules perform a variety of tasks. Some modules are hardware device drivers, such as the smc–ultra module for the SMC Ultra Ethernet card. Other modules provide support for the wide array of filesystems available in Linux, such as the ISO8859 filesystem used on CD–ROMs or the DOS FAT filesystem with long filename support (vfat).

Each entry in the listing produced by the lsmod command begins with the name of the module followed by the size of the module. As the size field indicates, modules are small. Often, they work together to get the job done. The interrelationships of modules are called *module dependencies*, which are an important part of properly managing modules. The listing tells you which modules depend on other modules. In our sample, the smc–ultra driver depends on the 8390 module. You can tell that from the 8390 entry, but not from the smc–ultra entry. The 8390 entry lists the modules that depend on it under the heading Used by.

Most of the lines in Listing 1.7 contain the word autoclean. This means that a module can be removed from memory automatically if it is unused. autoclean is only one of the module options. You can select different options when manually loading modules.

Manually Maintaining Modules

Modules can be manually loaded using the insmod command. This command is very straightforward—it's just the command and the module name. For example, to load the 3c509 device driver, enter **insmod 3c509**. This does not install the module with the autoclean option. If you want this driver removed from memory when it is not in use, add the –k option to the insmod command, and enter **insmod –k 3c509**.

One limitation with the insmod command is that it does not understand module dependencies. If you used it to load the smc–ultra module, it would not automatically load the required 8390 module. For this reason, modprobe is a better command for manually loading modules. As with the insmod command, the syntax is simple. To load the smc–ultra drive, simply enter **modprobe smc–ultra**.

modprobe reads the module dependencies file that is produced by the depmod command. Whenever the kernel or the module libraries are updated, run depmod to produce a new file containing the module dependencies. The command depmod –a searches all of the standard modules libraries and creates the necessary file. After it is run, you can use modprobe to install any module and have the other modules it depends on automatically installed.

Use the rmmod command to remove unneeded modules. Again, the syntax is simple; rmmod appletalk removes the appletalk driver from your system.

These manual maintenance commands have limited utility on a running system, because the correct things are usually done by Linux without any prodding from you. For example, I booted a small system on my home network, and immediately ran lsmod. I saw from this listing that I had appletalk and ipx installed, and I knew I didn't need either one. I typed in **rmmode appletalk**, but the message returned was rmmod: module appletalk not loaded because the system had already removed this unneeded module faster than I could type the command. Additionally, attempting to remove a command that is currently active returns the message Device or resource busy. For these reasons, I have rarely needed to use the rmmod command on an operational system.

In Sum

This chapter has taken a network server from power up to full operation. We have gone from the ROM BIOS to the Linux boot loader to the kernel initialization to the init process and, finally, to the boot scripts. All of these things play an important role in starting the system, and all of them can be configured by you.

Many operating systems hide the boot details, assuming that the administrator will be confused by the messages. Linux hides nothing. It accepts the fact that ultimately you're in control of this process, and you can exercise as much or as little of that control as you want. You can modify kernel behavior with boot prompt input, and control the behavior of the Linux loader through the lilo.conf file or the grub.conf file. You configure the init process through the inittab file and control system services through the startup scripts. All of these configuration files are text files that are completely under your control.

Other than the rc.local file, you will rarely change the files discussed in this chapter. But when you do need to fix or debug something, it is good to know where and when things happen in the boot process. Knowledge is a good thing, even if you only use it to ensure that your support contractors know what they are talking about.

An important piece of knowledge gained from this chapter is the understanding of how startup really works. Underneath all of the different tools provided by all of the different Linux distributions there is a boot process that has many similarities. Knowing where the files are stored that start and configure critical network services is very valuable information for any network administrator, particularly when things go wrong. In the next chapter, "The Network Interface," we look even deeper into the process that configures the server's network interface.

Chapter 2: The Network Interface

Overview

Nothing is more basic to network configuration than the interface the system uses to connect to the network. On most Linux servers, the network interface is an Ethernet card. Yet Linux systems are not limited to using Ethernet for network access. There are several types of network interfaces. One widely used network interface is the computer's serial port. Linux provides excellent support for serial–line communications, including a full range of tools to run TCP/IP over a serial line using Point–to–Point Protocol (PPP).

This chapter begins the discussion of configuring a Linux system as a network server by looking at how network interfaces are installed and configured. We begin with the Ethernet interface, which is the most popular TCP/IP network interface, and then go on to discuss how the serial interface is used for data communications. Finally, this chapter covers how PPP software is configured to turn the serial port into a TCP/IP network interface.

Configuring an Ethernet Interface

A Linux Ethernet interface is composed of both a hardware adapter card and a software driver. There are many possible brands and models of Ethernet cards. Select a card that is listed in "The Linux Hardware Compatibility HOWTO" by Patrick Reijnen, or listed among the approved hardware at your Linux vendor's website. When you find a card that works well for you, stick with it until you have a good reason to change.

Loadable Ethernet Drivers

The Ethernet interface software is a kernel driver. The driver can be compiled into the kernel or can be loaded as a loadable module, which is the most common way to install an Ethernet driver. On a Red Hat system, the loadable Ethernet drivers are found in the /lib/modules/*release*/kernel/drivers/net directory, in which *release* is the kernel version number. A directory listing of the network device drivers found on a Red Hat 7.2 system is shown in Listing 2.1.

Listing 2.1: Loadable Network Device Drivers

```
$ cd /lib/modules/2.4.7-10/kernel/drivers/net
$ ls *.o
3c501.o             at1700.o    eepro100.o  ne2k-pci.o     slhc.o
3c503.o             atp.o       eepro.o     ne3210.o       slip.o
3c505.o             bonding.o   eexpress.o  ne.o           smc-ultra32.o
3c507.o             bsd_comp.o  epic100.o   ni5010.o       smc-ultra.o
3c509.o             cs89x0.o    eql.o       ni52.o         starfire.o
3c515.o             de4x5.o     es3210.o    ni65.o         strip.o
3c59x.o             de600.o     eth16i.o    ns83820.o      sundance.o
8139too.o           de620.o     ethertap.o  pcnet32.o      sungem.o
82596.o             defxx.o     ewrk3.o     plip.o         sunhme.o
8390.o              depca.o     hamachi.o   ppp_async.o    tlan.o
ac3200.o            dgrs.o      hp100.o     ppp_deflate.o  tun.o
acenic.o            dl2k.o      hp.o        ppp_generic.o  via-rhine.o
aironet4500_card.o  dmfe.o      hp-plus.o   ppp_synctty.o  wavelan.o
aironet4500_core.o  dummy.o     lance.o     rcpci.o        wd.o
aironet4500_proc.o  e1000.o     lne390.o    sb1000.o       winbond-840.o
arlan.o             e100.o      lp486e.o    shaper.o       yellowfin.o
```

```
arlan-proc.o        e2100.o     natsemi.o   sis900.o
```

The loadable network device drivers available on this system are listed here. A few, such as ppp_ async.o and plip.o, are not for Ethernet devices. Most are easily identifiable as Ethernet drivers, such as the 3COM drivers, the SMC drivers, the NE2000 drivers, and the Ethernet Express drivers.

The Linux system detects the Ethernet hardware during the initial installation, and installs the appropriate driver. Normally, this is a completely automatic process that requires no input from the system administrator, but not always. Sometimes, Ethernet adapters are not detected by the initial installation. Other times, the adapter is added after the initial installation or an adapter has a non–standard configuration that must be communicated to the device driver. On rare occasions, the device driver itself is incorrect and needs to be replaced. When these things happen, users turn to you for help.

In general, when the system reboots newly installed hardware is configured by the hardware detection program provided by the Linux vendor. On Red Hat systems, the hardware–detection program is kudzu. It probes the system and creates the configuration file /etc/sysconfig/hwconf. The file is created by kudzu at the initial system startup. All subsequent runs of kudzu probe the system and compare the results to those found in the hwconf file. A new configuration is created only if a new piece of hardware is discovered that is not already in the hwconf file. Listing 2.2 is an excerpt of the hwconf file that contains an Ethernet card configuration.

Listing 2.2: An Ethernet Card Configuration Created by *kudzu*

```
-
class: NETWORK
bus: ISAPNP
detached: 0
device: eth
driver: smc-ultra
desc: "SMC EtherEZ (8416):Unknown"
deviceId: SMC8416
pdeviceId: SMC8416
native: 1
active: 0
cardnum: 0
logdev: 0
io: 0x200
dma: 0,0
-
```

Do not edit the hwconf file. Manually placing an entry in this file does not properly configure hardware that was not detected by kudzu. If newly installed hardware was not detected, place an empty file named reconfigSys in the /etc directory, as follows:

```
# touch /etc/reconfigSys
```

Then, reboot the system. This flag causes the Red Hat system to rerun the powerful anaconda hardware configuration program that is used during the initial Red Hat installation. Those readers with a Sun Solaris background are familiar with this technique. It is almost identical to the way a reconfiguration is forced on a Solaris system. Most systems have some technique for forcing a fresh probe of the hardware.

If the Ethernet adapter is not detected during the operating system installation, by kudzu, or by anaconda, you can manually load the device driver using the modprobe command described in Chapter 1, "The Boot Process." After the driver is installed, it must also be properly configured.

Configuring an Ethernet Device Driver

In most cases, the system correctly configures the network device driver without any help from the system administrator. Most drivers probe the card to discover the correct configuration. Additionally, the Ethernet drivers expect the adapters to use the manufacturer's default configuration, and if they do, no configuration changes are needed. But these techniques don't always work. When they don't, Ethernet adapter configuration parameters can be passed to the kernel through the boot prompt (as described in Chapter 1) for drivers that are compiled into the kernel. Optional configuration settings can be passed to loadable module Ethernet drivers using the insmod command. For example, the insmod command that tells the smc–ultra.o driver to use IRQ 10 and I/O port address 340 is

```
insmod smc-ultra.o io=0x340 irq=10
```

Additionally, most distributions provide tools to simplify setting the hardware configuration of Ethernet device drivers. Figure 2.1 shows you the Network Configuration tool that Red Hat provides. In Figure 2.1, we use the tool to configure the TCP/IP software. By selecting the Hardware tab, we could have used the Network Configuration tool to set the I/O port address and the IRQ for the Ethernet device driver.

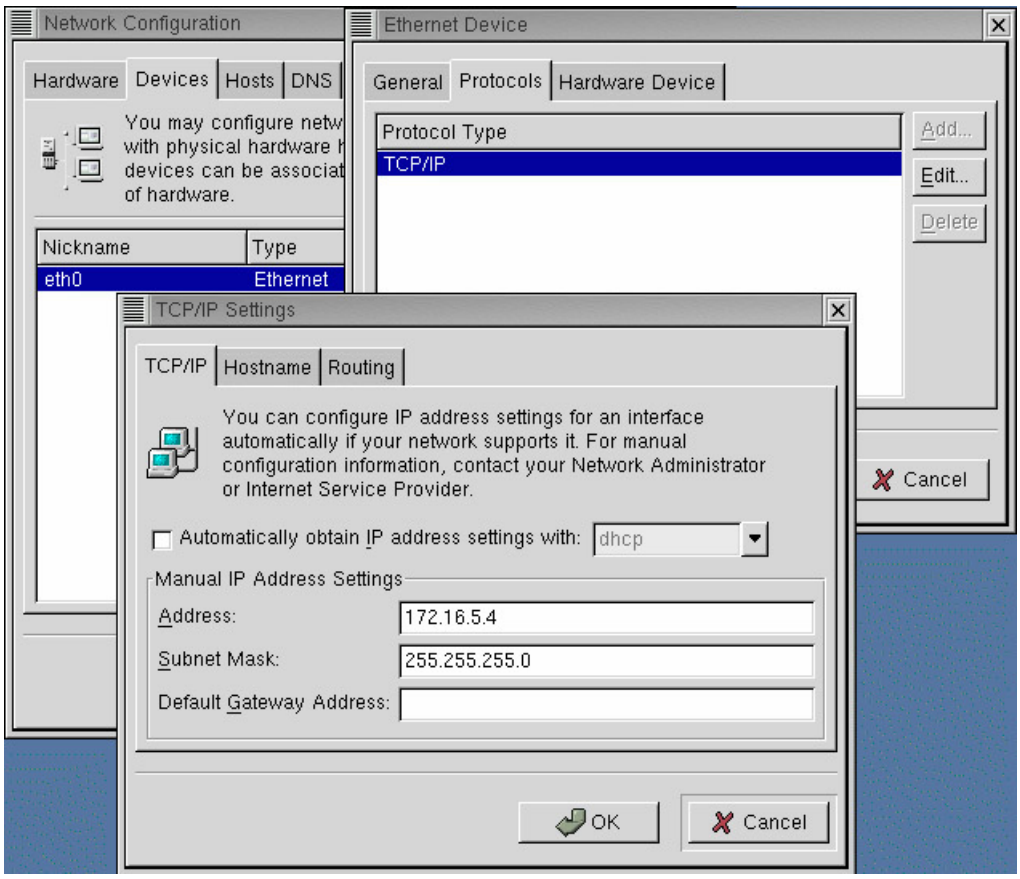


Figure 2.1: Red Hat's Network Configuration tool

It is not necessary to create a custom configuration for the driver if the card uses the manufacturer's default configuration, or if the driver can detect the correct configuration. Configuration conflicts are a problem only with older adapters. Manual hardware configuration is needed only in rare circumstances.

Compiling a New Device Driver

An even more rare problem for an Ethernet adapter is a bad or missing device driver. It is possible to obtain hardware that is so new there is no driver for the hardware incorporated in the Linux distribution. This, of course, breaks one of the fundamental rules for selecting the correct hardware for a server: never use hardware that is not listed in the Linux distribution vendor's hardware compatibility list. Of course, sometimes we do break the rules because we must have the very latest hardware, or because we do not have the freedom to choose our own hardware.

For a device driver to operate correctly, it must be compiled with the correct libraries for your kernel. Sometimes, this means downloading the driver source code and compiling it yourself on your system.

The source code for many Linux Ethernet drivers can be found at http://www.scyld.com/, which also has complete instructions for compiling each Ethernet driver. Many Ethernet drivers depend on other modules, such as pci_scan.c, which must also be downloaded. To simplify this, http://www.scyld.com/ also stores the driver source files in RPM Package Manager (SRPM) format. The RPM Package Manager is used repeatedly throughout this book as a way to simplify software installation.

After the adapter hardware and device driver are installed, the Ethernet interface can be used for a number of different network protocols. It can run NetWare protocols or, as described in Chapter 9, "File Sharing," it can run Server Message Block (SMB) protocol. Both of these are useful, but the primary network protocol used on Linux systems is TCP/IP. In the next section, we configure the Ethernet interface for TCP/IP.

The *ifconfig* Command

The ifconfig command assigns TCP/IP configuration values to network interfaces. Many values can be set with this command, but only a few are really needed: the IP address, the network mask, and the broadcast address. Assume that we have a network that uses the private network address 172.16.0.0 with the subnet mask 255.255.255.0. Further, assume that we need to configure a system named robin.foobird.org that is assigned the address 172.16.5.4. The ifconfig command to configure that interface is

```
ifconfig eth0 172.16.5.4 netmask 255.255.255.0 \
    broadcast 172.16.5.255
```

Note The network address 172.16.0.0, which is used as an example in this book, cannot be used to route data across the Internet. It is a private network number that is set aside for use on private networks.

The IP Address

The IP address is a software address specific to TCP/IP. Each device on the network has a unique address, even if the network is as large as the global Internet. In the previous example, the ifconfig command assigns the IP address 172.16.5.4 to the Ethernet interface eth0. You must define an IP address for every interface, either manually or through the use of a DHCP server, because the TCP/IP network is independent of the underlying hardware, which means that the IP address cannot be derived from the network hardware.

This approach has advantages and disadvantages, and is different from the address approach used by some other networks. NetBIOS uses the Ethernet hardware address as its address, and NetWare IPX incorporates the Ethernet address into the NetWare address. Using the address that

is available in the hardware makes these systems simple to configure because the system administrator does not need to be concerned with or knowledgeable about network addresses. But these systems are dependent on the underlying Ethernet, making it difficult or impossible to run them over global networks. TCP/IP is more difficult to configure, but it has the power to run a global network.

The *netmask* Argument The IP address includes a network portion that is used to route the packet through the Internet and a host portion that is used to deliver the packet to a computer when it reaches the destination network. The netmask argument identifies which bits in the IP address represent the network, and which bits represent the host. If no netmask (network mask) is defined, the address is divided according to the old address class rules. In effect, these rules say the following:

- If the first byte is less than 128, use the first eight bits for the network and the next 24 bits for the host.
- If the value of the first byte is from 128 to 191, use the first 16 bits for the network and the last 16 bits for the host.
- If the value of the first byte is from 192 to 223, use the first 24 bits for the network and the last eight bits for the host.
- Addresses with a first byte that is greater than 223 are not assigned to network hardware interfaces.

Except for the last one, these rules are used only if you fail to provide a netmask argument. Use netmask with the ifconfig command to define the address structure you want.

The old address classes did not provide enough flexibility for defining addresses. Three address classes proved inadequate to handle the huge number of addresses in the Internet and the incredible diversity of needs of the different networks connecting to the Internet. The solution is *classless IP addresses*. Classless addressing treats an IP address as 32 bits that can be divided between network and host portions in any way. The division of bits is controlled by a bit mask. If a bit is "on" in the mask (the bit is a one), the corresponding bit in the address is a network bit. If the bit is "off" (the bit is a zero), the corresponding bit in the address is a host bit. Here is our sample ifconfig command again:

```
ifconfig eth0 172.16.5.4 netmask 255.255.255.0 \
    broadcast 172.16.5.255
```

By the old class rules, the address 172.16.5.4 would define host 5.4 on network 172.16. The netmask argument 255.255.255.0 says that the first 24 bits of the address are the network portion, and that only the last eight bits are used to define the host. With this mask, the address is interpreted as host 4 on network 172.16.5.

Address Mask, Subnet Mask, or Network Mask?

These three terms, *address mask*, *subnet mask*, and *network mask*, are used interchangeably to refer to the same thing—the bit mask that is used to determine the structure of an address. Because the ifconfig command uses the keyword netmask for the argument that defines the bit mask on the command line, the ifconfig documentation refers to this value as the network mask. Therefore, the term *network mask* is widely used by system administrators.

All IP addresses have an associated bit mask because it is needed to implement classless IP addresses. If your organization purchased an official block of addresses from an outside agency such as ARIN, you received an official network number and an address mask to go with it. This

address/mask pair defines the total address space available to your organization. Sometimes, an organization will subdivide its official address space in order to create additional networks within that address structure. These additional networks are used to simplify management and routing, and are created by increasing the number of network bits in the address mask. Traditionally, network administrators call this *subnetting*, and sometimes they refer to the address mask as a *subnet mask*.

From the point of view of the individual network server, it makes no difference. The mask is defined in the same way on the ifconfig command, regardless of the name it is given—*address mask*, *subnet mask*, and *network mask* are all the same thing.

Even when the network you're working with uses an IP address that conforms to the class rules, don't allow the network mask to default to the class value. Always define the network mask on the command line. Addressing is too important to leave to chance; make sure you're in control of it. For the same reason, it is useful to define the broadcast address.

The Broadcast Address

The *broadcast address* is used to send a packet to every host on a network. The standard broadcast address is composed of the network address and a host address of 255. Given the ifconfig statement shown previously, the default broadcast address is 172.16.5.255. Using the IP address of 172.16.5.4 and the netmask of 255.255.255.0 gives a network address of 172.16.5.0. Add to that the host address of 255 to get 172.16.5.255. So why did I define the broadcast address instead of letting it default? Because you might be surprised by the default broadcast address.

You can check the configuration of an Ethernet interface by using the ifconfig command with only the interface name as a command–line argument. This does not change any configuration values; it displays the values that have already been set. Here is an example:

```
# ifconfig eth0 172.16.5.4 netmask 255.255.255.0
# ifconfig eth0
eth0 Link encap:Ethernet  HWaddr 00:60:97:90:37:51
     inet addr:172.16.5.4  Bcast:172.16.255.255
          Mask:255.255.255.0
     UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
     RX packets:319 errors:0 dropped:0 overruns:0
     TX packets:76 errors:0 dropped:0 overruns:0
     Interrupt:3 Base address:0x300
```

In this example, we configure the interface but do not define the broadcast address, expecting that the default will be exactly what we want. Instead, the default appears to ignore the netmask argument, and creates a broadcast address that would be correct if we used the old address class rules. Be specific. There are several files and scripts involved in setting the network interface configuration during the boot. Unless you're specific about it, you might not get the configuration you want.

Configuring the Interface for Every Boot

The configuration values assigned by the ifconfig command do not survive the boot. To configure the interface every time the system boots, the ifconfig command must be stored in a startup file. Normally, this does not require any effort on your part. Configuring the network interface is a basic part of the Linux installation.

During installation, Linux detects the network interface; and asks for the address, address mask, broadcast address, and several other network–related parameters. The installation program stores these values on the disk, where they are used later by the ifconfig command. Slackware stores the values in /etc/rc.d/rc.inet1; Caldera, Mandrake, and Red Hat store the values in /etc/sysconfig/network and /etc/sysconfig/network–scripts/ifcfg.*interface*, where *interface* is the name of the network interface, such as ifcfg.eth0. The startup scripts provided by these distributions then use the values to configure the interface.

However, if you want to manually configure the interface, you could directly configure it by storing the ifconfig command in the rc.local script. The rc.local script is the last startup script executed, so anything stored there overrides the configuration done by the system. The following commands placed in rc.local would configure the network interface on robin in exactly the manner we wanted:

```
ifconfig eth0 172.16.5.4 netmask 255.255.255.0 \
    broadcast 172.16.5.255
```

The ifconfig command provides the configuration described previously.

Many administrators don't edit any of the boot script files directly, nor do they manually configure the interface. Instead, they use a network configuration tool to correct any problems with the network interface configuration. Configuration tools are simple to use, but they are different on every Linux distribution, and they frequently change between releases of the same distribution. The ifconfig command is consistent. It works on every Linux distribution and every type of Unix.

Network Interface Configuration Tools

Most Linux distributions offer menu–driven or graphical–configuration tools for the network interface. Every distribution is different, but they all provide some tool. In this section, we used the Network Configuration tool provided with Red Hat 7.2.

On a Red Hat 7.2 system, the network interface is configured through the Network Configuration tool found on the Programs → System menu. The Network Configuration tool presents a window with four tabs:

Hardware Select the Hardware tab to add, remove, or configure a network adapter. To remove an adapter, highlight the adapter in the list presented on this tab, and click the button labeled Delete. To add an adapter, click the Add button and select the hardware type, which is either Ethernet, Modem, ISDN, or token ring. The Network Adapter Configuration window then appears. It has input boxes that accept an IRQ, the adapter memory address, the I/O port address, and the DMA request number. To configure an adapter, select the adapter from the list and click Edit. The same Network Adapter Configuration window used to add an adapter appears, giving you the chance to change the various hardware configuration values.

Devices Select the Devices tab to add, remove or configure a Linux network device. To remove a device, highlight the device in the list presented on this tab, and click the button labeled Delete. To add a device, either copy an existing device and edit the result, or click Add and select the device type. A device configuration window appropriate to the device type appears. For example, the Ethernet Device window has three tabs:

General Use this tab to enter the device name, for example, eth0, and to select whether or not the device should be started at boot time.

Protocols Use this tab to add, delete, or configure the network protocol associated with the device. TCP/IP is configured through this tab.

Hardware Device Use this tab to associate the device to a specific hardware adapter. For example, eth0 could be associated with the SMC Ultra Ethernet adapter.

Hosts Select the Hosts tab to make entries in the /etc/hosts table. The hosts file is covered in Chapter 4, "Linux Name Services."

DNS Select the DNS tab to define the Domain Name System configuration for a DNS client. Use the tab to set the system's hostname and domain name, and to provide the addresses of the name servers the system should use. DNS server configuration is covered in Chapter 4.

To use the Red Hat Network Configuration tool to enter the same network interface configuration that we created earlier with the ifconfig command; select Programs from the Start menu, System from the Programs menu, and Network Configuration from the System menu. In the Network Configuration window, select the Devices tab. On the Devices tab, highlight the eth0 device, and click Edit. In the Ethernet Device window, select the Protocols tab. On the Protocols tab, highlight TCP/IP, and click Edit. The result of this "point–and–click–fest" is shown in Figure 2.1.

The interface in the figure has already been configured. The fields and values are self–explanatory. Less the broadcast address, this is essentially the same configuration entered previously using the ifconfig command. Whether or not this is an easier way to enter the configuration values is a matter of personal opinion. In all Linux distributions, the designers of the configuration tools make some decisions about what is needed, where it should be defined, and how the interface should look. One of the great things about Linux is that if you disagree with the tool design, or if you want to do something differently, you can go directly to the commands that the tools really use to get the job done.

So far in this chapter, we have configured TCP/IP only on an Ethernet interface. This might lead you to believe that a Linux system requires TCP/IP and an Ethernet interface in order to communicate with other systems. That's not true. A Linux system can communicate without TCP/ IP, and it can be configured to run TCP/IP without an Ethernet interface. The next section looks at both capabilities.

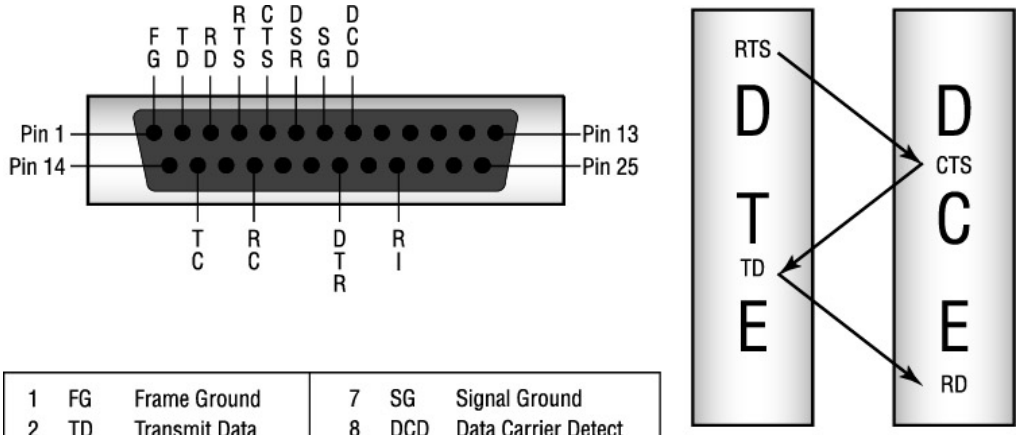
Note Clearly, you want to use TCP/IP, and you want to use your Ethernet interface. The features examined in the next section do not replace TCP/IP and Ethernet. Instead, they are additional capabilities that permit you to use the computer's serial interface in ways that would not be possible on some other network server systems.

The Serial Interface

Most PC hardware comes with two serial ports: a nine–pin connector or the traditional 25–pin RS–232 connector. In both cases, these connectors provide all of the signals needed to connect a terminal or a modem to the serial ports.

Figure 2.2 shows the RS–232 interface pin–out for the full 25–pin connector. The right side of the figure illustrates the electrical handshake that takes place when a PC communicates with a modem by showing the pins that are actually used during an exchange of data. In telecommunications talk,

the modem is called "data communication equipment" (DCE), and the PC is called "data terminal equipment" (DTE). The figure shows the exchange of signals between the DTE and the DCE.



1	FG	Frame Ground	7	SG	Signal Ground
2	TD	Transmit Data	8	DCD	Data Carrier Detect
3	RD	Receive Data	15	TC	Transmit Clock
4	RTS	Request To Send	17	RC	Receive Clock
5	CTS	Clear To Send	20	DTR	Data Terminal Ready
6	DSR	Data Set Ready	22	RI	Ring Indicator

Figure 2.2: The RS–232 hardware handshake

Only two of the interface pins are used to move data. Transmit Data (TD) is used to send data out of the computer, and Receive Data (RD) is used to read data into the computer. These are pins 2 and 3, respectively, on the RS–232 interface, as shown in the figure. A modem is able to directly connect to the computer because it reads data from the TD pin that the computer writes to, and writes data to the RD pin that the computer reads from.

A few other interface pins are used to set up and control the serial connection:

- Data Terminal Ready (DTR; pin 20) is used by the computer to signal that it is ready for the connection.
- Data Set Ready (DSR; pin 6) is used by the modem to signal that it is ready to connect.
- Data Carrier Detect (DCD; pin 8) is used by the modem to signal that it has a good connection to the computer at the remote end of the telephone line.
- Request To Send (RTS; pin 4) is used by the computer to indicate when it is ready to accept and send data.
- Clear To Send (CTS; pin 5) is used by the modem to indicate when it is ready to accept data.

Connecting a modem and a computer is straightforward because they use the interface pins in complementary ways. However, you'll have a conflict if you attempt to connect two computers together because they both want to use the same pins in the same way. For example, both will try to write to TD and read from RD. If you want to make a direct connection between two computers, buy a *null–modem cable*, also called a *direct connect cable* or a *cross–over cable*. The null–modem cable simply crosses some wires so that the two computers can communicate.

Connecting through the Serial Interface

Regardless of whether data comes through a modem or from a directly attached terminal, it is handled by the Linux system in the same way. Three programs handle the connection: init, getty, and login.

init is responsible for attaching the getty program to a serial port and for restarting the getty program whenever it terminates. You saw an example of this in the description of the inittab file in Chapter 1.

getty monitors the serial port. When getty detects a carrier signal on the port, it displays the login: prompt. It reads in the user's name and uses it to invoke login. For example if getty received norman in response to the login prompt, it would issue the command login norman.

login then prompts the user for a password, and checks the password using the appropriate authentication scheme. One of a number of schemes can be selected during the initial installation, as described in Appendix A, "Installing Linux." For example, the password might be checked against the encrypted password in the /etc/passwd or /etc/shadow file. The Linux user is given a few tries to enter the correct password. After the password is verified as correct, the UID and GID associated with the username are assigned to the tty device, and the following environment variables are set:

HOME This variable defines the user's home directory. login takes the value from the / etc/passwd file.

SHELL This variable defines the user's login shell. login takes the value from the /etc/ passwd file.

LOGNAME This variable defines the name by which the user is identified in the system log. login uses the username passed to it by getty.

PATH This variable defines the execution path. login defaults to a path of /usr/local/bin:/ bin:/usr/bin.

MAIL This variable defines the path to the user's mail file. login uses the path /var/mail/spool/*username*, where *username* is the name passed to login by getty.

TERM This variable identifies the terminal type. login keeps the TERM environment variable set by getty. The value of TERM will be a valid terminal type from the /etc/termcap file.

After these variables are set, login starts the shell identified in the user's /etc/passwd entry. The shell processes the initialization files that it finds in the user's home directory. These initialization files, such as .bash_profile and .bashrc for the bash shell, permit users to set their own environment variables; for example, to define a more complete execution path. Finally, the shell issues the command prompt to the user, and the user has access to the system.

All of these processes and services happen automatically and require almost no configuration on your part beyond creating a user account for the user and configuring the modem to answer the telephone. The basic Linux system comes with the capability to support terminal connections through serial ports. If you have ever configured dial–up services on other PC operating systems, you'll appreciate what an advantage this is.

Of course, even on a Linux system, things don't always work smoothly. Linux comes with some terminal–emulation programs such as seyon and minicom that can be used to troubleshoot modems and serial links.

Note See Chapter 13, "Troubleshooting," for information on troubleshooting with minicom.

The support for serial communications that is built into Linux is the foundation for running TCP/ IP

over a serial line. The next section covers just that.

Running TCP/IP Over a Serial Port

Of much greater utility than the capability to connect a terminal to your server's serial port is the capability to run TCP/IP over a serial port. Doing so allows you to run TCP/IP over a telephone line with a modem. This is, of course, the way that most people connect to the Internet through a local ISP. But it is also a way for you to provide connectivity into the Internet or your enterprise network for a remote field office or for users working at home.

Point–to–Point Protocol (PPP) provides the framing mechanism for sending IP datagrams over a telephone line. PPP uses a three–layered architecture to accomplish this:

> **Data Link layer** PPP uses a High–Level Data Link Control (HDLC) protocol to provide reliable data delivery over any type of serial line.
>
> **Link Control layer** A Link Control Protocol (LCP) was specifically developed for PPP. It opens and closes connections, monitors link quality, and negotiates the link configuration parameters.
>
> **Network Control layer** PPP is designed to carry a wide variety of network protocols. Protocols in this layer provide the control information that is necessary to customize the PPP link for the type of network traffic it is carrying. The network control protocol for TCP/IP is the Internet Protocol Control Protocol (IPCP).

Properly configuring a PPP service requires that all of these layers are correctly installed as well, and that the serial port and modem over which the traffic passes are properly configured. The remainder of this chapter looks at how these things are done on a Linux system.

Installing PPP

The layers of the Point–to–Point Protocol are implemented in Linux as a combination of kernel drivers and a PPP daemon (pppd). The HDLC Data Link layer protocol is installed as a kernel module, as are the Physical layer protocols for the serial devices upon which PPP depends.

The serial device drivers are usually compiled into the Linux kernel. Therefore, an external modem can be attached to a serial port without installing any additional device drivers. An internal modem, however, may require a special device driver. The Linux hardware detection software should detect the internal modem during the initial operating system installation or the first time the system is rebooted after a new internal modem is installed. Figure 2.3 shows kudzu detecting a newly installed internal modem.

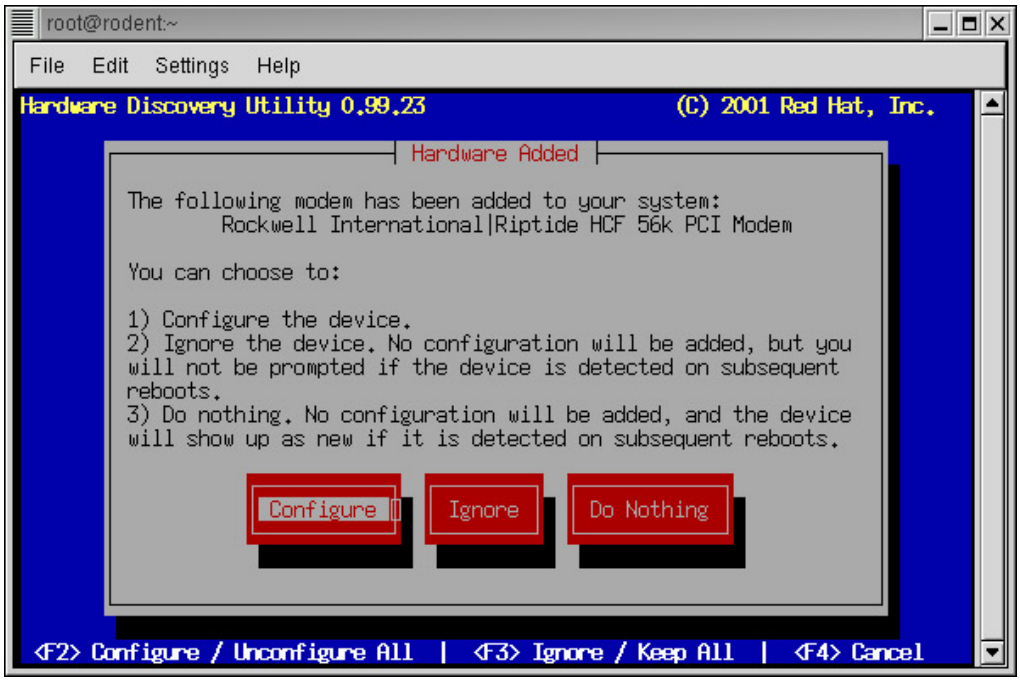


Figure 2.3: kudzu installing a modem driver

Selecting the Configure button installs the device driver module that provides the physical layer services for this modem. kudzu is a Red Hat tool, but all Linux distributions have some way to detect newly installed hardware.

Although the Physical layer and Data Link layer portions of PPP are provided by the kernel, the Link Control layer and the Network Control layer are provided by the PPP daemon. Thus, the lower layers of PPP are implemented as kernel modules, and the upper layers are implemented as a daemon process. In order for PPP to function, all of these components must be properly installed.

The PPP Kernel Module

Chapter 1 showed the kernel messages that are displayed when the serial drivers are installed. Similarly, when PPP is compiled into the kernel, messages about PPP are displayed during startup, as in this Caldera example:

```
$ dmesg | grep PPP
PPP: version 2.2.0 (dynamic channel allocation)
PPP Dynamic channel allocation code copyright 1995 Caldera, Inc.
PPP line discipline registered.
```

If PPP is installed by your kernel, you're ready to run the PPP daemon. On most systems, however, the kernel component of PPP is not compiled in the kernel. If it isn't, you must install the loadable module manually. To do so, use the commands covered in Chapter 1, and refer to the examples of installing an Ethernet driver earlier in this chapter. The modprobe command could be used to install the PPP kernel modules. For example, modprobe ppp_async would load the ppp_ async.o module, the ppp_generic.o modules upon which it depends, and the slhc.o module upon which ppp_generic depends. This would provide the kernel modules required by PPP.

As explained in Chapter 1, different Linux distributions use different tools for maintaining the modules list. All, however, provide the modprobe commands for you to specify that PPP modules should be included in the kernel at startup.

The PPP Daemon

The PPP daemon is started by the pppd command. The command can be entered at the shell prompt, and it often is on client computers. On server systems, the command is usually stored in a shell script to run at boot time for dedicated PPP connections or on demand for dial–up connections. Red Hat systems provide the /etc/sysconfig/network–scripts/ifup–ppp script to start the PPP daemon. However, the script is not edited directly. The values that control the ifup–ppp script are found in the ifcfg–ppp0 file in the same directory. Values can be placed in the ifcfg–ppp0 file through the same Network Configuration tool used earlier in this chapter for the Ethernet configuration.

Some administrators find it more convenient to create a custom pppd configuration using the standard pppd commands and configuration files. Many administrators find it just as simple as using a graphical configuration tool, and they like the fact that the commands and files are the same on all Linux distributions. The syntax of the pppd command is

```
pppd [tty-device] [speed] [options]
```

- *tty–device* is the name of the serial device over which the PPP protocol will operate. If no device is specified, the controlling terminal is the device that is used. As you'll see later, the ability to use the controlling terminal is very useful when creating a dial–up PPP server.
- *speed* is the transmission speed of the port, written in bits per second.
- *options* are just that—command–line options.

There are an enormous number of pppd options. In addition to specifying options on the command line, there are three different files available to store these options:

- /etc/ppp/options is used to store system–wide PPP options. This file is created and maintained by the system administrator.
- .ppprc, which each user can create and store in their home directory, is used to set personal PPP options.
- /etc/ppp/options.*device* sets PPP options for a specific serial device. For example, options.ttyS0 sets PPP options for /dev/ttyS0.

The files are read in the order listed previously, which means that options in the last file read can override options in the first file read. Thus, the order of precedence for options from all of these sources is as follows:

1. Options defined on the command line have the highest priority.
2. Options defined in the options.*device* file have the next priority.
3. Options defined by the user in the .ppprc file have the next priority.
4. Options from the /etc/ppp/options file have the lowest priority.

Looking at this list raises the concern that the system–wide options defined in the /etc/ppp/options file can be overridden by the user with the .ppprc file. Don't be overly concerned. Items that relate to system security cannot be overridden by the user. Additionally, you can always specify important options in the options.*device* file, which has a higher priority.

Note There are more than 70 options available for the pppd command. If you want to read about all of them, see *Using and Managing PPP*, by Andrew Sun (O'Reilly, 1999).

The following sections cover just those pppd options that you are most likely to use. By selecting the

correct options, you can configure pppd for a dedicated line or for a dial–up line as either a PPP server or as a client.

Configuring a PPP Server

A Linux system can be used as a PPP server for both dedicated connections and dial–up connections. Configuring pppd for a dedicated line is the simplest configuration, and it provides a good example of the structure of the pppd command. A single line inserted in the rc.local startup file is all that is necessary to configure a PPP server for a dedicated line:

```
pppd /dev/ttyS1 115700 crtscts
```

This command starts the PPP daemon, and attaches it to the serial device ttyS1. It sets the line speed for this dedicated line to 115700bps.

One option, crtscts, is also selected in this command. crtscts turns on Request To Send (RTS) and Clear To Send (CTS) hardware flow control. *Hardware flow control* uses the RTS and CTS pins in the serial interface to control the flow of data.

Always use hardware flow control with PPP. The alternative, software flow control, sends special characters in the data stream to control the flow of data. Software flow control, which is also called in–band flow control, at best wastes bandwidth doing something that could be done with hardware, and at worst sends control characters that can become confused with the actual data.

The pppd command for a client connected to a dedicated link would look the same as the preceding one, except that it would also have the defaultroute option:

```
pppd /dev/ttyS1 115700 crtscts defaultroute
```

defaultroute creates a default route that uses the remote PPP server as the default router. If a default route is already defined, this option is ignored. The defaultroute option is not used on the server end of the dedicated link because the server is the client's router, and therefore must already have another route to the outside world. defaultroute is used when the PPP link is the only link to the outside world, which is sometimes the case. This sample pppd command could be used to connect a small branch office into the enterprise network.

PPP configuration for a dedicated line is simple because there are always the same two systems connected to the line—one at each end, a single server and a single client. The line is dedicated to this single purpose and therefore can be configured at startup and left unchanged for as long as the system is running. There is no need to configure the server to handle multiple clients.

However, PPP clients and server are not always connected to dedicated serial lines. It is more common for them to be connected via dial–up serial lines, and configuring a server for dial–up lines is more complex than configuring it for dedicated lines.

PPP Dial–Up Server Configuration

There are three techniques for creating a dial–up PPP server. The key for two of them is the /etc/passwd file. One technique is to create a shell script, often named /etc/ppp/ppplogin, and use it as the login shell for dial–up PPP users, as in this example:

```
jane:x:522:100:Jane Resnick:/tmp:/etc/ppp/ppplogin
```

This looks exactly like any other /etc/passwd entry, and functions in exactly the same way. The PPP user is prompted for a username—jane in this case—and a password. After the user successfully logs in, she is assigned the home directory /tmp. The /tmp directory is commonly used for PPP users. The system then starts the user's login shell. In this case, the login shell is /etc/ppp/ ppplogin, which is actually a shell script that starts the PPP server. Here is a sample ppplogin script:

```
#!/bin/sh
mesg -n
stty -echo
exec /sbin/pppd crtscts modem passive auth
```

Your ppplogin script will not necessarily look like this example; you create your own ppplogin script. The mesg and stty commands are primarily to show you that you can put whatever you think is necessary in the ppplogin script. The mesg –n line prevents users from sending messages to this terminal with programs such as talk and write. Clearly, you don't want extraneous data being sent over the PPP connection.

The stty –echo command turns off character echo. When echo is on, the characters typed by the remote user are echoed back to the remote computer by the local computer. This was used on old Teletype terminals so that the user could monitor the quality of the dial–up line. If the characters were garbled as they appeared on the screen, the user knew that they should disconnect and redial to get a clear line. Of course, those days are long gone. Echoing characters across a PPP line is never used.

The real purpose of the script is, of course, to start the PPP daemon, and that is exactly what the last line does. There are definite differences between the pppd command that you execute here and the one that you saw in the previous section for dedicated lines. First, this command does not specify a device name. That's intentional. When pppd is started without a device name, it attaches to the controlling terminal, and runs in background mode. The controlling terminal is the terminal that login was servicing when it launched the ppplogin script. This permits you to use the same ppplogin script for every serial port. Likewise, this pppd command does not specify a line speed. In this case, the line speed is taken from the configuration of the serial port, again allowing you to use the same script for every serial port.

The remaining four items on the pppd command line are options:

- The crtscts option turns on hardware flow control, as discussed earlier.
- The modem option tells the PPP daemon to monitor the modem's Data Carrier Detect (DCD) indicator. By monitoring DCD, the local system can tell if the remote system drops the line. This is useful because it is not always possible for the remote system to gracefully close the connection.
- The passive option tells pppd to wait until it receives a valid Link Control Protocol (LCP) packet from the remote system. Normally, the PPP daemon attempts to initiate a connection by sending the appropriate LCP packets. If it doesn't receive a proper reply from the remote system, it drops the connection. Using passive gives the remote system time to initiate its own PPP daemon. With passive set, pppd holds the line open until the remote system sends an LCP packet.
- The auth option requires the remote system to authenticate itself. This is not the username and password authentication required by login, and it does not replace login security. PPP security is additional security designed to authenticate the user and the computer at the other end of the PPP connection.

An alternative to the ppplogin script is to use pppd as a login shell for dial–in PPP users. In this case, a modified /etc/passwd entry might contain

```
ed:wJxX.iPuPzg:101:100:Ed Oz:/etc/ppp:/usr/sbin/pppd
```

Here, the home directory is /etc/ppp and the login shell is the full path of the pppd program. When the server is started in this manner, server options are generally placed in the /etc/ppp/.ppprc file.

The final technique for running PPP as a server is to allow the user to start the server from the shell prompt. To do this, pppd must be installed setuid root, which is not the default installation. After pppd is setuid root, a user with a standard login account can log in and then issue the following command:

```
$ pppd proxyarp
```

This command starts the PPP daemon. After the client is authenticated, a proxy ARP entry for the client is placed in the server's ARP table so that the client appears to other systems to be located on the local network.

Of these three approaches, I prefer to create a shell script that is invoked by login as the user's login shell. With this approach, I don't have to install pppd setuid root. I don't have to place the burden of running pppd on the user. And I get all of the power of the pppd command plus all of the power of a shell script.

PPP Security

PPP has two authentication protocols: Password Authentication Protocol (PAP) and Challenge Handshake Authentication Protocol (CHAP). PAP is a simple password security system. CHAP is a more advanced system that uses encrypted strings and secret keys for authentication. Authentication helps to prevent intruders from accessing your server through its serial ports.

PAP Security

Password Authentication Protocol is vulnerable to all of the attacks of any reusable password system. PAP is better than no security, but not by much. PAP sends the PPP the client name and the password as clear text at the beginning of the connection setup. After this initial authentication, the client is not reauthenticated. Although spying on a serial line is much more difficult than spying on an Ethernet, PAP clear–text passwords can still be stolen by someone spying on your network traffic. Additionally, an established session can be hijacked by a system spoofing addresses.

Because of these weaknesses, use PAP only when you must—for example, if you have to support a client that can only provide PAP authentication. Unfortunately, PAP is still very widely used, and may be your only choice.

To configure PAP, make appropriate password entries in the /etc/ppp/pap–secrets file. A pap–secrets file might contain the following:

Listing 2.3: A Sample pap–secrets File

```
# Secrets for authentication using PAP
# client    server  secret              IP addresses
crow        wren    Wherearethestrong?  172.16.5.5
wren        crow    Whoarethetrusted?   172.16.5.1
```

Given the configuration shown in Listing 2.3, crow sends the PPP client name crow and the password Wherearethestrong? when asked for authentication by wren. wren sends the client name wren and the password Whoarethetrusted? when asked for authentication by crow. Both systems have the same entries in their pap−secrets files. These two entries provide authentication for both ends of the PPP connection.

The IP address field at the end of each entry defines the address from which the client name and the password are valid. Thus, only the host at address 172.16.5.5 can use the client name crow and the password Wherearethestrong?. Even though this is a valid client name and password combination, if it comes from any other address, it will be rejected.

The auth option on the pppd command line forces the PPP daemon to require authentication. If it must, it will fall back to PAP, but first it will try to use CHAP.

CHAP Security

Challenge Handshake Authentication Protocol is the default authentication protocol used by PPP. CHAP is not vulnerable to the security attacks that threaten PAP. In fact, a PPP connection that uses CHAP is probably more secure than your local Ethernet connection. For one, CHAP does not send clear−text passwords. Instead, CHAP sends a string of characters called a *challenge string*. The system seeking authentication encrypts the challenge string with a secret key from the /etc/ppp/chap−secrets file, and returns the encrypted string back to the servers. The secret key never travels across the network and therefore cannot be read off the network by a snooper.

Additionally, CHAP repeatedly reauthenticates the systems. Even if a thief steals the connection through address spoofing, he cannot keep the connection for long without responding correctly to the CHAP challenge.

CHAP is configured through the chap−secrets file. Entries in the chap−secrets file contain the following fields:

respondent This is the name of the computer that will respond to the CHAP challenge. Most documentation calls this the "client" field. However, PPP clients require authentication from servers in the same way that servers require authentication from clients. The first field defines the system that must respond to the challenge in order to be authenticated.

challenger This is the name of the system that will issue the CHAP challenge. Most documentation calls this the "server" field, but as noted earlier, servers are not the only systems that issue CHAP challenges. The second field contains the name of the computer that challenges the other system to authenticate itself.

secret This is the secret key that is used to encrypt and decrypt the challenge string. The challenger sends a challenge string to the system that is being authenticated. The respondent encrypts that string using the secret key, and sends the encrypted string back to the challenger. Then, the challenger decrypts the string with the secret key. If the decrypted string matches the original challenge string, the responding system is authenticated. Using this system, the secret key never travels across the network.

address This is an address written either as a numeric IP address or as a

hostname. If an address is defined, the respondent must use the specified IP address. Even if a system responds with the correct secret key, it will not be authenticated unless it is also the host at the correct IP address.

Listing 2.4 shows the entries that a chap–secrets file on robin might contain:

Listing 2.4: A Sample chap–secrets File

```
# cat chap-secrets
# Secrets for authentication using CHAP
# client    server  secret              IP addresses
robin       wren    Peopledon'tknowyou  robin.foobirds.org
wren        robin   ,andtrustisajoke.   wren.foobirds.org
```

When robin is challenged by wren, it uses the secret key Peopledon'tknowyou to encrypt the challenge string. When robin challenges wren, it expects wren to use the secret key ,andtrustisajoke.. It is very common for entries to come in pairs like this. After all, there are two ends to a PPP connection, and both systems require authentication to create a secure link. wren challenges robin, and robin challenges wren. When both computers are sure they are communicating with the correct remote system, the link is established. For this to work, of course, wren needs the same entries in its chap–secrets file.

For security reasons, it is very important to protect the /etc/ppp directory. Only the root user should be able to read or write the chap–secrets file or the pap–secrets file. Otherwise, the secret keys may be compromised. Additionally, only the root user should be allowed to write the options file. Otherwise, users would be able to define system–wide PPP options.

Finally, only the root user should be able to write to the script files ip–up and ip–down. pppd runs the ip–up script as soon as it makes the PPP connection, and it runs the ip–down script after it closes the connection. These scripts can perform privileged functions relating to the network connection. Thus, allowing anyone but the root user to modify these scripts compromises the security of your system.

PPP Client Configuration

Configuring a PPP client is as complex as configuring a server. The primary reason for this complexity is the fact that the client initiates the PPP connection. To do that, the client must be able to dial the server's phone number and perform any necessary login procedures. A pppd command for a client system might look like this:

```
pppd /dev/cua1 115700 connect "chat -v dial-server" \
     crtscts modem defaultroute
```

You have seen all but one of these options before. In fact, this command is almost identical to the PPP client configuration we created earlier for the dedicated link except for the connect option. The connect option identifies the command used to set up the serial–line connection. In the sample, the command is enclosed in quotes because it contains whitespace characters. The complete command is chat –v dial–server.

The chat program is used to communicate with devices, such as modems, attached to a serial port. The –v option causes chat to log debugging information through syslogd. In the example,

dial−server is the name of the script file that chat uses to control its interaction with the modem and the remote server.

chat Scripts

A chat script defines the steps that are necessary to successfully connect to a remote server. The script is a list of expect/send pairs. Each pair consists of a string that the local system expects to receive, separated by whitespace from the response that it will send when the expected string is received. A sample script might contain the following:

Listing 2.5: A Sample chat Script

```
$ cat dial-server
' ' ATZ
OK ATDT301-555-1234
CONNECT \d\d\r
gin: sophie
ord: TOga!toGA
```

The script in Listing 2.5 contains instructions for the modem as well as the login for the remote server. The first line, expects nothing, which is what the empty string (' ') means, and sends a reset command to the modem. (ATZ is the standard Hayes reset command.) Next, the script expects the modem to send the string OK, and it responds with a Hayes dial command (ATDT). When the sample modem successfully connects to the remote modem, it displays the message CONNECT. In response to this, the script waits two seconds (\d\d) and then sends a carriage−return (\r).

Most systems don't really require anything like this, but it provides an example of a chat escape sequence. chat provides several escape sequences that can be used in the expect string or the response string. Table 2.1 lists these sequences and their meanings.

Table 2.1: Escape Sequences and Their Meanings

Escape Sequence	Meaning
\b	The backspace character
\c	Don't send a terminating carriage−return; used at the end of a send string
\d	Delay for one second
\K	A line break
\n	A newline character
\N	An ASCII null character
\p	Pause for 1/10 of a second
\q	Send the string, but don't record it in the log; used at the end of a send string
\r	A carriage−return
\s	The space character
\t	The tab character
\\	The backslash
ddd	The ASCII character with the octal value *ddd* (for example, *177* is the DEL character)
^character	A control character (for example, *^G* is a Ctrl+G)

The last two lines of the sample script are the remote login. The script expects gin:, which are the last four characters of the login: prompt, and responds with the username sophie. Next, ord:, which are the last four characters of the Password: prompt is expected, and TOga!toGA is sent as a response. Once the login is complete, the remote server runs the ppplogin script, and the PPP connection is up and running.

Note chat is a very elementary scripting language. It is popular for setting up PPP connections because most PPP connections do not require a complex script. If yours does, you may need to use a more powerful scripting language. Linux provides both dip and expect. To read more about dip, see *TCP/IP Network Administration*, by Craig Hunt (O'Reilly, 2002). To read more about expect, see *Exploring Expect*, by Don Libes (O'Reilly, 1997).

Using an X Tool to Configure a PPP Client

So far, we have configured PPP by editing the configuration files with a text editor. It is also possible to configure a PPP client by using a graphical tool running under X Windows. Every Linux distribution offers at least one tool for this purpose, and a new set of tools is released with every new version of Linux. Red Hat 7.2 alone offers three easily accessible tools to do this one task. Perhaps the most accessible tool is the one that is launched by double−clicking the Dialup Configuration icon located on the GNOME desktop. This starts an installation wizard.

The first time Dialup Configuration runs, it automatically detects the modem attached to the system, if none has been configured. It then moves on to configuring a dial−up connection for the modem, asking for the phone number and then the user name and password. Finally, in a window labeled Other Options appears, in which you are encouraged to select Normal ISP from a drop−down menu. Normal ISP sets the normal defaults used for a PPP connection. These default values can be adjusted later.

Subsequent runs of Dialup Configuration displays a window labeled Internet Connections. This window contains two tabs: Accounts and Modems. Select the Accounts tab to add new dial−up scripts, delete old ones, or edit the dial−up and PPP characteristics of an existing script. Select Modems to add a new modem, delete an existing modem, or edit the characteristics of an existing modem. Figure 2.4 shows the Internet Connections window and the Edit Internet Connections window that appears when a current connection in the Internet Connections window is highlighted and the Edit button is clicked.

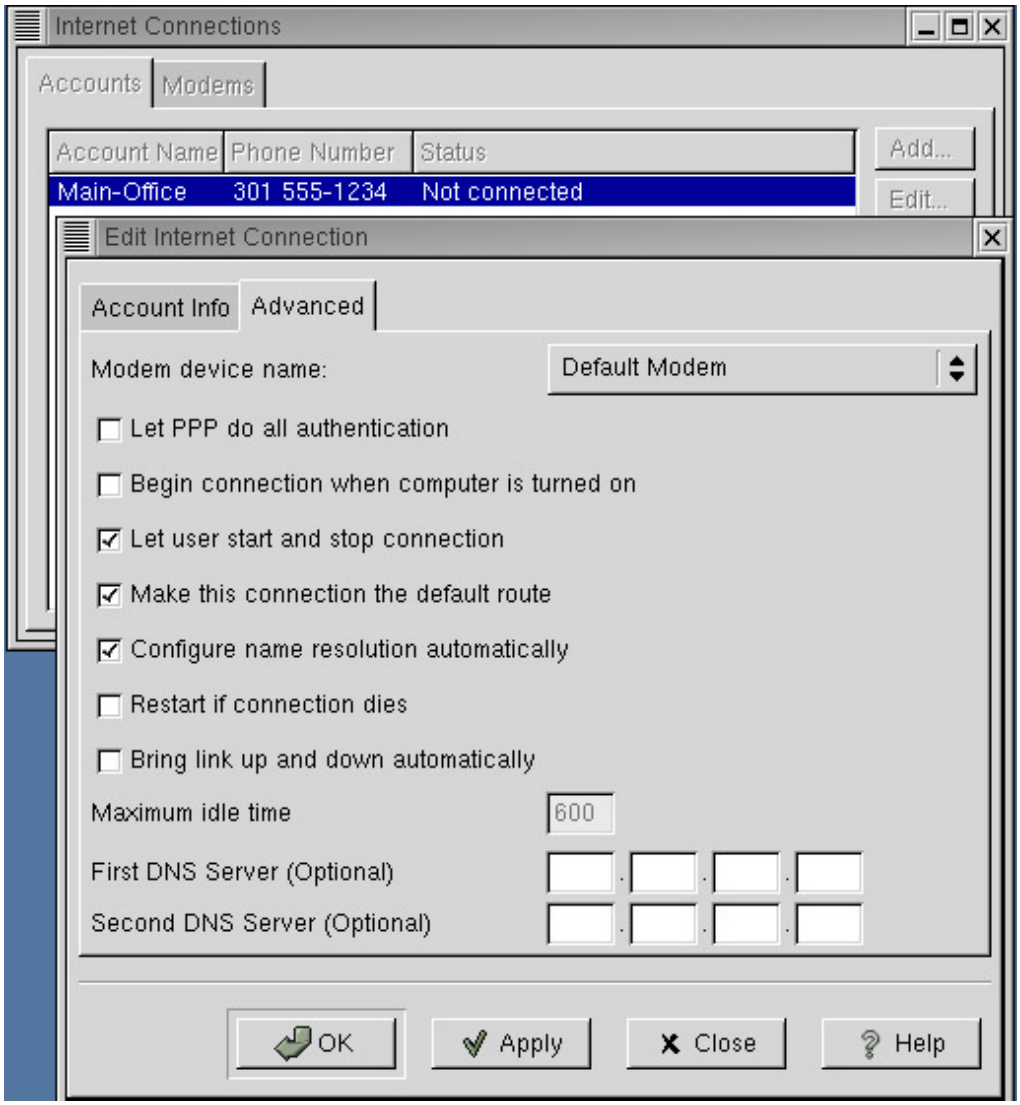


Figure 2.4: The Internet Connections window

The Edit Internet Connections window has two tabs. The Account Info tab defines the username, password, and telephone number used to establish the connection. This tab also contains the Account Name field. The Account Name is any arbitrary label that identifies this connection to the user. In the figure, we used the account name Main–Office because this account is used to connect a branch office into the main office.

The second tab is label Advanced. The Advanced tab is used to set various PPP parameters. The following actions set configuration values through this tab:

- Select the modem used for this connection from the drop–down menu.
- Use the Let PPP Do All Authentication check box to set the PPP auth parameter described earlier.
- Use the Make This Connection The Default Route check box to set the PPP defaultroute parameter described earlier.
- Use the Begin Connection When the Computer Is Turned On check box to start the PPP connection at boot time. Use this setting when the client always connects to the same server and when the client should be "always on." This is the setting you would probably use for a branch office connecting into headquarters. Two other settings provide alternative ways to control when the connection is made.
- Use the Let The User Start The Connection check box to permit the user to manually make

the connection before using any TCP/IP services that depend on the connection. Use this setting on clients that use multiple servers. To make the connection, the user opens the Internet Connections window, highlights the desired connection, and clicks the Debug (or Dial) button. There is one final way that connections are made.

- Use the Bring Link Up And Down Automatically check box to allow applications to make the connection on demand. When this check box is selected, launching the Mozilla browser causes the client to dial the server. The connection is brought down when it is idle for the amount of time specified in the Maximum Idle Time box.
- The tab offers two techniques for configuring name service. Check the Configure Name Resolution Automatically box if the configuration is provided externally. Enter the IP addresses of two DNS servers in the boxes at the bottom of the tab if the DNS configuration is being provided here.

Deciding whether or not to use tools such as Dialup Configuration to set up a PPP client is mostly a matter of personal taste. Some people prefer the graphic interface, whereas others do not. Either way, the end result is the same, and the information provided is the same as well.

In Sum

A network server requires a network interface. On most servers, including Linux systems, this is an Ethernet interface. However, Linux systems also support a full range of network services through the serial port. Connect a terminal or modem to a Linux serial port, and it will support user logins. Run the Point–to–Point Protocol (PPP), and the serial port can be used to provide TCP/IP connections.

The same flexibility that Linux shows in the types of network connections it provides is seen in the full range of network services offered by a Linux system. In the next chapter, we start configuring some of these services with the basic Internet services: telnet and ftp.

Part II: Internet Server Configuration

Chapter List

Chapter 3: Login Services
Chapter 4: Linux Name Services
Chapter 5: Configuring a Mail Server
Chapter 6: The Apache Web Server
Chapter 7: Network Gateway Services

Part Overview

Featuring:

- Configuring inetd and xinetd
- Managing user accounts
- Configuring the ftpaccess file
- Understanding the Domain Name System (DNS)
- Configuring the BIND stub resolver and the Lightweight Resolver
- Creating a DNS server with BIND
- Managing a name server with ndc and rndc
- Understanding the sendmail.cf file
- Configuring sendmail with m4
- Configuring Apache with the httpd.conf file
- Configuring SSL and web server access controls
- Understanding Apache log files
- Configuring routing with route, routed, and gated
- Configuring a network address translation server

Chapter 3: Login Services

Overview

At this point, Linux is installed and the network interface is running. Now we begin configuring network services for the users. This chapter covers two traditional services that have been part of TCP/IP networking since the beginning: Telnet and FTP.

The Telnet daemon (telnetd) allows users to log in and run any available applications directly on a Linux server. Some would say this turns the Linux system into a *compute server* because the remote system has access to the computer power of the server. However, the term compute server is confusing because most people think it relates solely to scientific, number–crunching applications. Others would say that remote login access turns the Linux server into a terminal server because the remote systems act like terminals connected to the server. However, this terminology is also confusing because many people think of the hardware adapters that increase the number of terminals that can connect to a server when they hear the words *terminal server*. In this book, we say that services such as Telnet turn the Linux server into a network login server because it allows a user to log in and run applications directly on the Linux system. Telnet extends to the TCP/IP network the same login service that was described for serial ports in Chapter 2, "The Network Interface."

The other service covered in this chapter is the File Transfer Protocol (FTP). It allows users to transfer files to and from the server. Like Telnet, FTP requires a user to log in before using the service.

Telnet and FTP services are usually installed during the initial Linux installation. The system administrator then needs to ensure the following:

- The server daemons are started when they are needed
- The users have valid accounts to log in to the servers

Properly performing both of these tasks to configure network login services is the topic of this chapter.

FTP requires the two configuration steps described previously, but it can require additional configuration in order to provide special services, such as anonymous FTP. (*Anonymous FTP* allows people who don't have a valid user account to log in to the system.) This chapter covers the configuration of optional FTP services with examples drawn from Red Hat Linux.

Note Security issues surround both Telnet and FTP. See Chapter 12, "Security," for ways to secure these services.

The chapter begins by looking at how telnetd, ftpd, and various other network services are started when they are needed. Two techniques for starting services on–demand are examined: the Extended Internet Services daemon (xinetd) used by Red Hat and the Internet Services daemon (inetd) used by several other Linux distributions. We start with the fundamentals of why and how services are started on–demand.

Starting Services On–Demand

Network services are started in two ways: at startup by a boot script or on–demand. Chapter 1, "The Boot Process," discusses scripts and tools that are used to control which services are started at boot time. Listing 1.6 shows the large number of boot scripts used to start services, many of which are easily identified by name as starting network services.

Despite the large number of services started by the boot scripts, many network services are started on–demand by inetd, or alternatively by xinetd. Network services that are started at boot time continue to run, whether or not they are needed, but inetd and xinetd start services only when they are actually needed.

Each startup technique has its own advantages. Starting services on–demand saves the resources that are used when an unneeded service is left running. On the other hand, starting a daemon at boot time saves the overhead associated with repeated startups for a service that is in constant demand.

Depending on which one your system uses, either inetd or xinetd is started at boot time, and continues to run in the background as long as the system is running. The daemon listens to the network ports and starts the appropriate service when data arrive on the port associated with the service. In the same way that getty detects traffic on a serial port and starts login to handle a terminal connection, inetd and xinetd detect traffic on the network, and start the proper service to handle that traffic. (Don't remember getty or login? See Chapter 2.) To understand this process, you need to understand a little about TCP/IP ports.

Protocol and Port Numbers

Data travel through a TCP/IP network in packets called datagrams. Each datagram is individually addressed with the following:

- The IP address of the host to which it should be delivered
- The protocol number of the transport protocol that should handle the packet after it is delivered to the host
- The port number of the service for which the data in the packet are bound

For data to be delivered correctly on a global scale, as it is on the Internet, the IP address must be globally unique, and the meaning of the protocol and port numbers must be well–known to all systems in the network. (Chapter 2 describes how the IP address is assigned to the network interface during the installation.) In the case of the IP address, you're the one responsible for making sure that it is unique. (In Chapter 13, "Troubleshooting," you'll see what kinds of problems occur when the IP address isn't unique.) Protocol and port numbers are different; they are defined by Internet standards. Thus, the protocol numbers and port numbers can be predefined in two files, /etc/protocols and /etc/services, that come with the Linux system.

The */etc/protocols* File

Data from the network arrive at the computer as one stream. The stream may contain data packets from multiple sources bound for multiple applications. In telecommunications terminology, we say that the data stream is *multiplexed*. To deliver each packet to the correct application, it must be *demultiplexed*. The first step in this process is for the Internet Protocol to pass the packet to the correct transport protocol. IP determines the correct protocol by means of the protocol number that is contained in the datagram packet header.

The /etc/protocols file identifies the protocol number of each transport protocol. Listing 3.1 is an excerpt from the protocols file from a Red Hat system.

Listing 3.1: An Excerpt of the */etc/protocols* File

```
$ head -33 /etc/protocols
# /etc/protocols:
# $Id: protocols,v 1.3 2001/07/07 07:07:15 nalin Exp $
#
# Internet (IP) protocols
#
#       from: @(#)protocols     5.1 (Berkeley) 4/17/89
#
# Updated for NetBSD based on RFC 1340, Assigned Numbers (July 1992).
#
# See also http://www.iana.org/assignments/protocol-numbers

ip      0       IP              # internet protocol, pseudo protocol number
#hopopt 0       HOPOPT          # hop-by-hop options for ipv6
icmp    1       ICMP            # internet control message protocol
igmp    2       IGMP            # internet group management protocol
ggp     3       GGP             # gateway-gateway protocol
ipencap 4       IP-ENCAP        # IP encapsulated in IP (officially "IP")
st      5       ST              # ST datagram mode
tcp     6       TCP             # transmission control protocol
cbt     7       CBT             # CBT, Tony Ballardie
                                <A.Ballardie@cs.ucl.ac.uk>
egp     8       EGP             # exterior gateway protocol
igp     9       IGP             # any private interior gateway (Cisco: for
                                IGRP)
bbn-rcc 10      BBN-RCC-MON     # BBN RCC Monitoring
nvp     11      NVP-II          # Network Voice Protocol
pup     12      PUP             # PARC universal packet protocol
argus   13      ARGUS           # ARGUS
emcon   14      EMCON           # EMCON
xnet    15      XNET            # Cross Net Debugger
chaos   16      CHAOS           # Chaos
udp     17      UDP             # user datagram protocol
mux     18      MUX             # Multiplexing protocol
dcn     19      DCN-MEAS        # DCN Measurement Subsystems
hmp     20      HMP             # host monitoring protocol
```

/etc/protocols is a simple text file. Lines that begin with a sharp sign (#) are comments. Active entries begin with the protocol name, followed by the protocol number, and optionally by alternate names for the protocol and by a descriptive comment. The comment is often helpful for identifying the protocol. Frequently, the alternate name of a protocol is simply the standard name in uppercase letters, but this is not always the case. Look at protocol numbers 4, 10, 11, and 19. These alternate names are more than just a change of case. However, protocol names are not as important as protocol numbers. The protocol number is contained in the header of the datagram, and it is the number that is used for data delivery.

Other than cosmetic differences, the protocols file on your Linux system, regardless of the distribution, will look similar to the excerpt shown in Listing 3.1. In fact, you can find a similar file on any Unix or Windows NT system because the protocol numbers are standardized. You will *never* need to edit this file.

The /etc/protocols file on a Red Hat 7.2 system contains about 150 lines. Despite the size of this

file, only two entries are significant for most network services. One, tcp, defines the protocol number for Transmission Control Protocol—the TCP in TCP/IP. Its protocol number is 6. The other, udp, defines the protocol number for User Datagram Protocol as 17. Several of the entries in this file define protocol numbers used by routing protocols. Many other entries define experimental protocols that are not widely used. TCP and UDP carry most of the information you are interested in.

The */etc/services* File

The second stage of demultiplexing the network data is to identify the application to which the data are addressed. The transport protocol does this using the port number from the transport protocol header.

The standard port numbers are identified in the /etc/services file. The port numbers for well–known services are assigned in Internet standards, so you never change the port number of an existing service. On rare occasions, you may need to add a new service to the file, but that is the only time you would edit this file.

Although there are many transport protocols, there is an even larger number of network services. For that reason, Listing 3.2 is only a small piece of the complete /etc/services file found on a Red Hat system.

Listing 3.2: An Excerpt from */etc/services*

```
ftp             21/tcp
ftp-data        20/udp
ftp             21/tcp
ftp             21/udp
ssh             22/tcp                          # SSH Remote Login Protocol
ssh             22/udp                          # SSH Remote Login Protocol
telnet          23/tcp
telnet          23/udp
# 24 - private mail system
smtp            25/tcp          mail
smtp            25/udp          mail
time            37/tcp          timserver
time            37/udp          timserver
rlp             39/tcp          resource        # resource location
rlp             39/udp          resource        # resource location
nameserver      42/tcp          name            # IEN 116
nameserver      42/udp          name            # IEN 116
nicname         43/tcp          whois
nicname         43/udp          whois
tacacs          49/tcp                          # Login Host Protocol (TACACS)
tacacs          49/udp                          # Login Host Protocol (TACACS)
re-mail-ck      50/tcp                          # Remote Mail Checking
                                                Protocol
re-mail-ck      50/udp                          # Remote Mail Checking
                                                Protocol
domain          53/tcp          nameserver      # name-domain server
domain          53/udp          nameserver
whois++         63/tcp
whois++         63/udp
bootps          67/tcp                          # BOOTP server
bootps          67/udp
bootpc          68/tcp                          # BOOTP client
bootpc          68/udp
tftp            69/tcp
```

```
tftp            69/udp
gopher          70/tcp                          # Internet Gopher
gopher          70/udp                             ftp-data        20/tcp
```

/etc/services has a format that is very similar to the format used by /etc/protocols. Comments begin with a sharp sign (#). Active entries begin with the name of the service, which is followed by a port number/protocol name pair, and optionally by an alternate name for the service and by a comment. The port number/protocol name pair defines the port number for the service, and identifies the protocol over which the service runs. The protocol name must be a valid name defined in /etc/protocols.

From Listing 3.2, you can tell that telnet uses port 23 and runs on top of the TCP transport protocol. Furthermore, you can tell that Domain Name System (DNS), referred to as domain in the file, uses port 53 on both TCP and UDP. Each transport protocol has a complete set of port numbers, so a single port number, such as 53, can be assigned to a service for both UDP and TCP. In fact, it is possible to assign a port number under UDP to one service and the same port number under TCP to a completely different service; however, in order to avoid confusion, this is never done.

inetd and xinetd can monitor any port listed in the /etc/services file. Which protocol and port numbers are monitored by these daemons are defined in the /etc/inetd.conf file for inetd or the /etc/xinetd.conf file for xinetd. These files, in turn, define which services are started by the daemons. We begin by looking at the inetd configuration.

Configuring *inetd*

The inetd configuration is defined in the /etc/inetd.conf file. The file defines the ports that inetd monitors and the pathnames of the processes it starts when it detects network traffic on a port. Many Linux systems use inetd. In fact, prior to version 7.0, Red Hat used inetd instead of xinetd. Listing 3.3, which shows the active entries in an inetd.conf file, was generated on a server running Red Hat 6.2.

Listing 3.3: Excerpts from an *inetd.conf* File

```
$ grep -v '^#' /etc/inetd.conf
ftp       stream  tcp nowait  root   /usr/sbin/tcpd      in.ftpd -l -a
telnet    stream  tcp nowait  root   /usr/sbin/tcpd      in.telnetd
shell     stream  tcp nowait  root   /usr/sbin/tcpd      in.rshd
login     stream  tcp nowait  root   /usr/sbin/tcpd      in.rlogind
talk      dgram   udp wait    root   /usr/sbin/tcpd      in.talkd
ntalk     dgram   udp wait    root   /usr/sbin/tcpd      in.ntalkd
imap      stream  tcp nowait  root   /usr/sbin/tcpd      imapd
finger    stream  tcp nowait  root   /usr/sbin/tcpd      in.fingerd
auth      stream  tcp nowait  nobody /usr/sbin/in.identd in.identd -l -e -o
linuxconf stream  tcp wait    root   /bin/linuxconf      linuxconf -http
```

Every entry in the inetd.conf file shown in Listing 3.3 defines a service that is started by inetd. Each entry is composed of seven fields:

Name This is the name of the service, as listed in the /etc/services file. This name maps to the port number of the service.

Type This is the type of data delivery service used. There are only two common

types: dgram for the datagram service provided by UDP, and stream for the byte stream service provided by TCP.

Protocol This is the name of the protocol, as defined in the /etc/protocols file. The name maps to a protocol number. All of the sample entries have either tcp or udp in this field.

Wait–status This is either wait or nowait. wait tells inetd to wait for the server to release the port before listening for more requests. nowait tells inetd to immediately begin listening for more connection requests on the port. wait is normally used for UDP, and nowait is normally used for TCP.

UID This is the username under which the service is run. Normally, this is root, but for security reasons, some processes run under the user ID nobody.

Server This is either the pathname of the server program that inetd launches to provide the service or the keyword internal, which is used for simple services that are provided by inetd itself. In Listing 3.3, the entries for linuxconf and auth are good examples. Both of these entries show the full path to the appropriate server program. All of the other entries in Listing 3.3, however, share the same server path: /usr/sbin/tcpd. Clearly, this is not really the path to the ftp server, the telnet server, and every other server. In reality, tcpd is a security feature used by Linux. tcpd is called the TCP Wrapper, and it is used to wrap security protection around network services. The details of how to use TCP Wrapper to improve security are covered in Chapter 12. For now, it is sufficient to know that tcpd will start the correct server when it is called by inetd.

Arguments These are the command–line arguments that are passed to the server program. The first argument is always the name of the server program being executed. The argument list looks exactly as the command would look if it were being typed in at a shell prompt.

As you can see in Listing 3.3, inetd.conf comes preconfigured with several services active. The Red Hat Linux 6.2 default inetd.conf file comes with an equal number of services inactive, as shown in Listing 3.4.

Listing 3.4: Services Disabled by *inetd*

```
$ grep '^#[a-z]' /etc/inetd.conf
#echo    stream  tcp     nowait   root    internal
#echo    dgram   udp     wait     root    internal
#discard stream  tcp     nowait   root    internal
#discard dgram   udp     wait     root    internal
#daytime stream  tcp     nowait   root    internal
#daytime dgram   udp     wait     root    internal
#chargen stream  tcp     nowait   root    internal
#chargen dgram   udp     wait     root    internal
#time    stream  tcp     nowait   root    internal
#time    dgram   udp     wait     root    internal
#exec    stream  tcp     nowait   root    /usr/sbin/tcpd     in.rexecd
#comsat  dgram   udp     wait     root    /usr/sbin/tcpd     in.comsat
#dtalk   stream  tcp     wait     nobody  /usr/sbin/tcpd     in.dtalkd
#pop-2   stream  tcp     nowait   root    /usr/sbin/tcpd     ipop2d
#pop-3   stream  tcp  nowait      root    /usr/sbin/tcpd     ipop3d
#uucp    stream  tcp  nowait       uucp    /usr/sbin/tcpd /usr/lib/uucp/
```

```
   uucico -l
#tftp    dgram   udp  wait        root    /usr/sbin/tcpd     in.tftpd
#bootps  dgram   udp  wait        root    /usr/sbin/tcpd     bootpd
#cfinger stream  tcp  nowait      root    /usr/sbin/tcpd     in.cfingerd
#systat  stream  tcp  nowait      guest   /usr/sbin/tcpd     /bin/ps -auwwx
#netstat stream  tcp  nowait      guest   /usr/sbin/tcpd     /bin/netstat
   -f inet
#swat    stream  tcp  nowait.400  root    /usr/sbin/swat     swat
```

Comments in the inetd.conf file begin with a sharp sign (#). To disable a service, insert a sharp sign at the beginning of its entry. To enable a service, remove the sharp sign. For example, enable the BootP server by removing the sharp sign at the beginning of the bootps entry. Likewise, to disable the finger protocol, insert a sharp sign before the first character in the finger entry. This simple edit gives you complete control over the services provided by your Linux system.

Disabling unneeded services is an important part of server security. Do not run services that you don't really use. Every network service is a potential hole for a security cracker to slither through. On our sample server, we offer ftp. Clearly, it should not be commented out of the sample configuration. But our server does not offer gopher or pop–3 services. Those services could be commented out without harming the users of this imaginary server. On another server, the roles might be reversed: POP might be enabled, and FTP might be disabled. The services that are enabled are driven by which services will be offered by the server.

The importance of controlling access to network services as a component of overall system security is the main reason that xinetd was created. The Extended Internet Services daemon (xinetd) performs the same task as inetd, but it offers enhanced security features.

Configuring *xinetd*

An alternative to inetd is the Extended Internet Services Daemon (xinetd). xinetd is configured in the /etc/xinetd.conf file, which provides the same information to xinetd as inetd.conf provides to inetd. But instead of using positional parameters with meanings determined by their relative location on a configuration line, xinetd.conf uses attribute and value pairs. The attribute name clearly identifies the purpose of each parameter. The value configures the attribute. For example, the third field in an inetd.conf entry contains the name of the transport protocol. In an xinetd.conf file, the name of the transport protocol is defined using the protocol attribute, for example, protocol = tcp. However, xinetd can do much more than inetd, so there are many more attributes available to configure xinetd than the seven positional parameters used by inetd. The best way to understand the various xinetd attributes is to look at some realistic configurations. Listing 3.5 is the xinetd.conf file from a Red Hat 7.2 system.

Listing 3.5: The *xinetd.conf* File

```
$ cat /etc/xinetd.conf
#
# Simple configuration file for xinetd
#
# Some defaults, and include /etc/xinetd.d/

defaults
{
        instances               = 60
        log_type                = SYSLOG authpriv
        log_on_success          = HOST PID
        log_on_failure          = HOST
```

```
        cps                     = 25 30
}

includedir /etc/xinetd.d
```

The lines that begin with # are comments. The defaults statement is the first active entry in this file. The defaults statement is optional, but when it is used, only one defaults statement can appear in the configuration. Use this statement to set default values for various attributes. Enclose the list of attribute/value pairs in curly braces. (Lists of attribute/value pairs are always enclosed in curly braces.) The xinetd.conf file in Listing 3.5 defines default values for five different attributes:

instances This specifies the maximum number of daemons providing any one type of service that can be running simultaneously. Listing 3.5 sets instances to 60, meaning that no more than 60 telnetd processes will be running at any one time. xinetd will not start the sixty–first instance, even if an additional service request for Telnet is received from the network. The system default for instances is to allow an unlimited number of simultaneous processes for all types of services. instances is set to a numeric value or to the keyword UNLIMITED.

log_type This defines where messages will be logged. There are two possible values for this attribute:

FILE *pathname* [*soft_limit* [*hard_limit*]] The FILE setting tells xinetd to log activity to the file identified by *pathname*. Upper limits for the size of the file are set using *soft_limit* and *hard_limit*. When *soft_limit* is exceeded, xinetd issues warning messages about the file size. When *hard_limit* is exceeded, xinetd stops logging messages to the file. The default is to have no limit on the size of the log file.

SYSLOG *syslog_facility* [*syslog_level*] The SYSLOG setting tells xinetd to use the standard syslogd daemon to log activity. *syslog_facility* must be set to one of the facilities defined in /etc/syslog.conf. *syslog_level*, if used, must be a standard syslogd severity value. The example in Listing 3.5 tells xinetd to log activity through syslogd and to use the authpriv facility.

log_on_success This defines the information that is logged when a successful connection is made to a local service. The xinetd.conf file in Listing 3.5 logs the address of the remote client (HOST) and the process ID of the daemon started to service that client (PID). There are several other values that can be logged. Properly configuring xinetd to log network activity is covered in detail in Chapter 12.

log_on_failure This defines the information that is logged when an attempt to connect to a local service is unsuccessful. The xinetd.conf file in Listing 3.5 logs the address of the remote system (HOST) that attempted the connection. Properly configuring xinetd to log network activity is an important security topic covered in detail in Chapter 12.

cps This sets connections–per–second limits for the services. The first value is the maximum number of connection for any one service that will be accepted in a single second. If that number is exceeded, xinetd will stop accepting connections for that service, and will wait the number of seconds specified by the second number before

accepting any more connections for that service. The values in Listing 3.5 tell xinetd to accept no more than 25 connections to any one service in a single second, and to wait 30 seconds if more than 25 connection attempts are made before accepting more connections for that service. cps prevents a service from taking all of the systems resources.

Besides the defaults statement, the only other statement in the Red Hat xinetd.conf file is includedir. The includedir statement includes by reference all of the files found in the specified directory. In Listing 3.5, the specified directory is /etc/xinetd.d, which is the directory used most often for this purpose. A single file can be included in the xinetd.conf configuration by using the include *filename* command, where *filename* is the name of a single file. However, the includedir statement is more powerful because it includes all of the files from an entire directory. This makes it possible to maintain a separate configuration file for each service. This sounds complicated, but in practice it is simple. The individual configuration files are very short, and because every item in a file applies to only one service, the purpose and meaning of each item is easier to understand. A listing of the /etc/xinetd.d directory shows the xinetd configuration files on our sample Red Hat system.

```
$ ls /etc/xinetd.d
chargen     echo      imaps         ntalk  rsh     talk  time-udp
chargen-udp echo-udp  ipop2         pop3s  rsync   telnet wu-ftpd
daytime     finger    ipop3         rexec  sgi_fam tftp
daytime-udp imap      linuxconf-web rlogin swat    time
```

The files in /etc/xinetd.d represent all of the services that can be started by xinetd on this system. Other network services are available from this server because they are started by boot scripts, but xinetd can only start a service for which it has a configuration file. Adding a completely new, on–demand network service to this system would require adding a new configuration file to the xinetd.d directory.

Most of the filenames in the /etc/xinetd.d directory clearly map to service names from the /etc/services file, but not all. The wu–ftpd file starts the FTP service, although wu–ftpd is not a standard service name. In this case, wu–ftpd stands for Washington University FTP Daemon (WU–FTPD), which is the version of the FTP server that is used by default with Red Hat 7.2. The wu–ftpd file is a good example of what most xinetd configuration files look like. Listing 3.6 shows the wu–ftpd file from a Red Hat 7.2 system.

Listing 3.6: The */etc/xinetd.d/wu–ftpd* File

```
$ cat /etc/xinetd.d/wu-ftpd
# default: on
# description: The wu-ftpd FTP server serves FTP connections. It uses \
#       normal, unencrypted usernames and passwords for authentication.
service ftp
{
        socket_type             = stream
        wait                    = no
        user                    = root
        server                  = /usr/sbin/in.ftpd
        server_args             = -l -a
        log_on_success          += DURATION USERID
        log_on_failure          += USERID
        nice                    = 10
        disable                 = no
}
```

Again, lines that begin with # are comments. This file, like most of the files in /etc/xinetd.d, contains just a single service statement. The format of a service statement is:

service *service_name*

{

attribute_list

}

The *service_name* is the official name of the service from the /etc/services file. The *service_name* must map to a port number. In Listing 3.6, the *service_name* is ftp, which maps to port number 21. (Notice in Listing 3.6 that the official service name is ftp, not wu–ftpd.) The *attribute_list*, just as it was in the defaults statement, is a list of attribute/value pairs enclosed in curly braces. The ftp service has nine attributes defined in Listing 3.6:

socket_type Specifies the type of socket used for this service. This is usually either dgram for the datagram service provided by UDP, or stream for the byte stream service provided by TCP. There are two other possible values: raw for a service that connects directly to IP, and seqpacket for sequential datagram delivery. raw and seqpacket are not used for standard services. Every xinetd configuration file in the xinetd.d directory on our sample Red Hat system sets socket_type to either dgram or stream.

wait Tells xinetd whether it should wait for the service to release the port before listening for more connections to that service. yes means wait, and no means don't wait. Most often, dgram sockets require xinetd to wait, and stream sockets permit xinetd to proceed without waiting.

user Defines the username used to run the service. Normally, this is root, but for security reasons, some processes run under the username nobody.

server This is the path to the program that xinetd should start when it detects activity on the port. In Listing 3.6, /usr/sbin/in.ftpd is the program that is started when a connection request arrives on port 21.

server_args These are the command–line arguments that are passed to the server program when it is started. Listing 3.6 shows two command–line options, both of which are specific to the Washington University FTP daemon. The –l option tells the daemon to log all transactions. The –a option tells it to read its configuration from the ftpaccess file. (More on the ftpaccess file later in this chapter.)

log_on_success This defines the information that is logged when a successful connection is made to the FTP service. Notice the += syntax used with the attribute. += adds the values defined here to the values previously defined for log_on_success in the defaults statement of the xinetd.conf file. (See Listing 3.5.) If = had been used instead of +=, the log_on_ success values defined in the defaults statement would have been replaced by these values for the FTP service. Instead, xinetd will combine both attribute settings and log the client's address (HOST), the process ID of the service (PID), the amount of time the client is connected to the server (DURATION), and the username used to log in to the server (USERID). xinetd logging is covered

again in Chapter 12.

log_on_failure This defines the information that is logged when an unsuccessful attempt to connect to the FTP service is made. Again, the += syntax is used to add the values defined here to the values previously defined for log_on_failure in the defaults statement of the xinetd.conf file. (See Listing 3.5.) These settings cause xinetd to log the client's address, and the username used for the failed login. xinetd logging is covered again in Chapter 12.

nice Defines the nice value that xinetd uses when it launches the server program. The nice command sets the scheduling priority for server program. The nice command, along with many other common system administration tools, is covered in *Linux System Administration*, by Vicki Stanfield and Rod Smith (Sybex, 2001).

disable Tells xinetd whether or not this service has been disabled by the system administrator. If disable is set to no, the service is enabled and xinetd runs this service when it receives a connection request on this server's port. If disable is set to yes, the service is disabled and xinetd does not start the service in response to connection requests. By default, services are enabled if the disable attribute is not included in the configuration.

This last attribute shows that a system administrator can control the services started by xinetd by changing the value of the disable attribute. This can be done by directly editing the configuration files found in the xinetd.d directory, yet Red Hat provides an easier way to do this. The chkconfig command used in Chapter 2 to control boot scripts can also be used to control services started by xinetd.

Listing 3.7: Using *chkconfig* to Control *xinetd*

```
[root]# chkconfig --list wu-ftpd
wu-ftpd         on
[root]# grep disable /etc/xinetd.d/wu-ftpd
        disable                 = no
[root]# chkconfig wu-ftpd off
[root]# chkconfig --list wu-ftpd
wu-ftpd         off
[root]# grep disable /etc/xinetd.d/wu-ftpd
        disable = yes
```

The first two commands in Listing 3.7 show that chkconfig believes that the FTP service is ready to run and that the disable attribute in the wu–ftpd configuration file is set to no. Then, the **chkconfig wu–ftpd off** command is entered to disable the FTP service. Another quick check with chkconfig and grep shows that FTP is now turned off, and the disable attribute in the wu–ftpd file is set to yes. Use the configuration filename (wu–ftpd, in this case), not the service name (ftp) when using the chkconfig command to enable or disable a service started by xinetd.

Now that you know how telnetd and ftpd are started, let's look at how the user accounts they require are created. The next section reviews the variety of tools that are used to create user accounts on Linux systems.

Creating User Accounts

Each user who logs in to a Linux system is identified by a user account. The user account controls access to the system by defining the username and password that authenticate the user during the login. After the user logs in to the system, the user account's user identifier (UID) and group identifier (GID) are used to control the user's privileges. These values, which are defined when creating a user account, control filesystem security and identify which users control what processes.

The user account is an essential part of a Linux system, and all Linux distributions provide tools for maintaining user accounts. This section examines several of these tools, but it first takes a look behind the scenes to see what these tools are doing for you.

The Steps to Creating a User Account

The tools that different Linux distributions offer to simplify the process of adding a user account may vary, but all of the tools ask for essentially the same information because the underlying process of adding a user account is the same on all Linux systems. Adding a user account requires the following steps:

1. Edit the /etc/passwd file to define the username, UID, GID, home directory, and login shell for the user.
2. Run a tool, such as passwd, to create an encrypted password for the user.
3. Run mkdir to create the user's new home directory.
4. Copy the default initialization files from /etc/skel to the user's home directory. The /etc/skel directory holds files such as .bashrc, which is used to initialize the bash environment. A Linux system comes with a selection of files already in /etc/skel. To provide additional or different initialization files for your users, simply add files to the /etc/skel directory, or edit the files that you find there.
5. Change the ownership of the user's home directory and the files it contains so that the user has full access to all of her files. For example, chown –r kathy:users /home/kathy.

Most of these steps involve building the user's home directory. However, much of the information about the user account is stored in the /etc/passwd file.

The *passwd* File

Every user on a Linux system has an entry in the /etc/passwd file. To see what accounts exist on a Linux system, just look inside that file. Listing 3.8 is the passwd file from our sample Red Hat system.

Listing 3.8: A Sample */etc/passwd* File

```
# cat /etc/passwd
root:gvFVXCMgxYxFw:0:0:root:/root:/bin/bash
bin:*:1:1:bin:/bin:
daemon:*:2:2:daemon:/sbin:
adm:*:3:4:adm:/var/adm:
lp:*:4:7:lp:/var/spool/lpd:
sync:*:5:0:sync:/sbin:/bin/sync
shutdown:*:6:0:shutdown:/sbin:/sbin/shutdown
halt:*:7:0:halt:/sbin:/sbin/halt
mail:*:8:12:mail:/var/spool/mail:
```

```
news:*:9:13:news:/var/spool/news:
uucp:*:10:14:uucp:/var/spool/uucp:
operator:*:11:0:operator:/root:
games:*:12:100:games:/usr/games:
gopher:*:13:30:gopher:/usr/lib/gopher-data:
ftp:*:14:50:FTP User:/home/ftp:
nobody:*:99:99:Nobody:/:
craig:6VKY34PUexqs:500:100:Craig Hunt:/home/craig:/bin/bash
sara:niuh3ghdj73bd:501:100:Sara Henson:/home/sara:/bin/bash
kathy:wv1zqw:502:100:Kathy McCafferty:/home/kathy:/bin/bash
david:94fddtUexqs:503:100:David Craig:/home/david:/bin/bash
becky:tyebwo8bei:500:100:Rebecca Hunt:/home/becky:/bin/bash
```

Most of these accounts are included in the passwd file as part of the initial installation; only the last five entries in Listing 3.8 are real user accounts added by the system administrator. The first entry is the root account for the system administrator, but most of the others are special accounts created for programs that need to control processes or that need to create and remove files.

Each /etc/passwd entry follows the *user*:*password*:*UID*:*GID*:*comment*:*home*:*shell* format, where

- *user* is the username. It should be no more than eight characters long, and should not contain capital letters or special characters. kristin is a good username.
- *password* is the encrypted password of the user. Of course, you don't actually type an encrypted password here. The encrypted password is stored here by the *passwd* command. If you use the shadow password file, and most Linux distributions do, the encrypted password will not actually appear here. Instead, the password will be stored in */etc/shadow*. See Chapter 12 for a description of the shadow password file.
- *UID* is the numeric user ID for this user account.
- *GID* is the numeric group ID of the primary group of this user.
- *comment* is text information about the user. At a minimum, you should have the user's first and last names. Some people like to include the user's telephone number and office room number. For historic reasons, this field is sometimes called the GECOS field.
- *home* is the user's home directory.
- *shell* is the login shell for this user.

Almost all of the information needed to create a user account appears in the passwd file. The next few sections examine some of this information in more detail.

Selecting a Login Shell

A *login shell* (or command shell) processes the command lines that are entered by the user. Linux provides a selection of several different login shells. Several shells are included in the distribution, and many more are available from the Internet. Despite the variety of shells, most sites standardize on one or two; every user account added to the sample *passwd* file in Listing 3.8 uses /bin/bash as the login shell.

The /etc/shells file is a list of valid shell names that is consulted by a number of programs to determine what shells are available on the system. Listing 3.9 is the /etc/shells file on a Red Hat system.

Listing 3.9: Available Login Shells

```
$ cat /etc/shells
/bin/bash2
```

```
/bin/bash
/bin/sh
/bin/ash
/bin/bsh
/bin/tcsh
/bin/csh
```

The /etc/shells file on our sample Red Hat system provides the following login shell selections:

> **/bin/sh** This is the Bourne Shell, which is the original Unix shell. The Bourne Shell introduced many of the fundamental concepts of command shells, but as you can imagine given the great age of Unix, the original Bourne Shell is seriously out–of–date. Despite that, feel free to make this selection on a Red Hat system because there is no Bourne Shell stored at /bin/sh. Instead, it is a link to /bin/bash.
>
> **/bin/bash** This is the Bourne–Again Shell, which is the most popular shell on Linux systems. bash is the Bourne Shell with all of the modern enhancements such as command–line editing, command history, and filename completion that were introduced by newer shell programs.
>
> **/bin/bash2** This is yet another version of the Bourne–Again Shell, which provides all of the features of bash. On the Red Hat system, bash2 is exactly the same as bash because /bin/ bash2 is just a logical link to /bin/bash.
>
> **/bin/ash** This is the A Shell. ash is a very compact program—only about one–fifth the size of bash. ash has minimal features in keeping with its very small size.
>
> **/bin/bsh** This is the B Shell, which is another minimal shell designed to provide basic features in a small–sized package. bsh is just a link to ash on a Red Hat system.
>
> **/bin/csh** This is the C Shell. csh is an early Unix shell with a command and scripting environment inspired by the C programming language. Though the original csh is now out–of–date, csh introduced important concepts, such as command histories, which are still used today. On a Red Hat system, /bin/csh is a link to /bin/tcsh.
>
> **/bin/tcsh** This is the Tenex C Shell, which is the enhanced csh. tcsh adds filename completion and command–line editing to the C shell.

This list of shells includes four shells that are just logical links, another one that is a minimal shell, and only two shells that are widely used as login shells: bash and tcsh. Other Linux distributions have other lists, and you can add the names of other shells to the /etc/shells file if you add other shells to your system. If you have users coming to a Linux environment from other Unix operating systems, they may demand the Korn Shell. Two versions of the Korn Shell are in widespread use:

> **/bin/ksh** This is the Korn Shell, which is one of the most popular Unix shells, and is the one that first introduced command–line editing.
>
> **/bin/zsh** The Z Shell closely resembles the Korn Shell, and it provides advanced features, such as command completion and built–in spell–checking.

The /etc/shells file provides default values for a number of programs. Keep in mind that the list is

just a suggestion; you don't have to select a shell from the list. You can type the pathname of any shell installed on your system as the user's login shell. There was an example of this in Chapter 2, when we used pppd as a login shell to configure PPP server accounts.

Understanding the User ID

The UID field is a unique numeric identifier for the user. The range of UID numbers on most Unix systems is 0 to 65536. On Linux systems using the Linux 2.4 kernel, the range is 0 to 4294967295. Numbers below 100 are reserved for special system accounts, such as uucp, news, mail, and so on. By definition, the root account is UID 0. Other than these restrictions, you can select any available number in the valid range.

Every user account has a UID and at least one GID. Every file and process on a Linux system also has a UID and a GID. Matching UIDs determine ownership of files and processes. Matching GIDs determine group access to files and processes.

On an isolated system, files are only available to users of that system. But on a network, files are available between systems through file sharing. The most popular file–sharing technique on Unix and Linux systems is Network File System (NFS). NFS uses the same file security mechanisms as the Linux system—the UID and the GID—and can work only if the user IDs and group IDs assigned on the various systems on the network are coordinated. For example, if tyler was assigned UID 505 on crow, and daniel was assigned 505 on robin, a potential conflict could exist. Mounting a filesystem from crow on robin would give daniel ownership privileges to files that really belonged to tyler! Because of this, care must be taken to develop a plan for assigning user IDs and group IDs across every system on your network.

> **Note** NFS and the issue of properly assigning user IDs and group IDs in an NFS environment are covered in Chapter 9, "File Sharing."

Understanding the Group ID

The GID field is used to identify the primary group to which the new user belongs. When a Linux system is first installed, several groups are included, most of which are either administrative groups, such as adm and daemon, or groups belonging to specific services, such as news and mail. users is a catch–all group for all users.

When you use an administrative tool to create a user account, the tool assigns a group for the user if you don't select one. On some systems, such as Slackware, the tool defaults to the group users. On a Red Hat system, the default is to create a brand–new group that contains only the new user as a member. Neither approach is exactly right for all cases.

The group ID, like the user ID, is used for filesystem security. On Linux systems, there are three levels of file permissions: ownership privileges, group privileges, and world privileges. If everyone is included in the same group, as is the case when everyone is placed in the users group, then everyone has the same group privileges when attempting to access anyone's files. In effect, group privileges are no different from world privileges. This defeats the purpose of the group ID, which is to allow groups to share files while protecting those files from people who are not in the group.

Likewise, if a group is created that contains only one user, the purpose of the group ID is defeated—there is no point in having group privileges if there is no group. The owner of a file already has access privileges for the file based on the UID, so the GID is unnecessary when the group is one. Using this approach, group privileges are no different from ownership privileges.

To make the most effective use of group IDs, you need to create groups. Develop a plan for the group structure you will use on your network. This plan doesn't need to be complicated. Most network administrators use an organizational group structure in which people in the same work group are members of the same GID. A more complex structure, based on projects, is also possible. Be careful, however, not to create a structure that requires lots of maintenance. Projects come and go, and you don't want to get into a situation in which you are constantly changing groups and moving files for users.

Note For full NFS support, the group structure plan needs to be coordinated among the systems on your network. See Chapter 9 for information on planning and coordinating a group structure.

Creating New Groups To create a group, add an entry for the new group in the /etc/group file. Every group has one entry in the file, and all of the entries have the same format, *name*:*password*:*gid*:*users*, where

- *name* is the name of the group.
- *password* is not usually used. Leave it blank, or fill it with a placeholder such as x.
- *gid* is the numeric group identifier. It is a number between 0 and 65536. GID 0 is used for the root group. Most administrators reserve the numbers below 100 for special groups.
- *users* is a comma–separated list of users assigned to this group. The primary group of a user is assigned in the /etc/passwd file. /etc/group assigns supplemental groups to a user.

Some examples from the /etc/group file on a Red Hat system illustrate this structure.

Listing 3.10: Examples from the */etc/group* File

```
root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
mail:x:12:mail
news:x:13:news
uucp:x:14:uucp
users:x:100:kathy
popusers:x:45:kathy
slipusers:x:46:
kathy:x:501:
```

In an example later in this chapter (refer to "Tools to Create User Accounts"), we create the user account kathy, and allow the system to create a default GID for kathy. By default, Red Hat creates a new group for the user using the username as the group name and using the first available number above 500 as the GID. That's where the kathy entry at the end of this file came from. Additionally, we edited the /etc/group file to grant kathy membership in the users and the popusers groups. That's why kathy appears in the user list of both of those entries. Note that kathy is not in the user list of the group kathy. That is because it is her primary group, which is assigned in the /etc/passwd file. Therefore, her primary group is kathy, and her supplemental groups are users and popusers. She is granted the group privileges of all three of these groups.

You can create a new group or modify an existing group by directly editing the /etc/groups file. Alternatively, you can create a group by using the tools provided by your Linux distribution. Use the groupadd command for this purpose. For example, to create a group for the sales department with a group name of sales and a GID of 890, enter **groupadd –g 890 sales**. To add a new group, simply select an unused group name, and an available GID number, and enter them into the /etc/ group file using the groupadd command.

The name or numeric GID of an existing group can be changed with the groupmod command. For example, to change the GID assigned to the sales group created above from 890 to 980, enter **groupmod –g 980 sales**. To change the group name from sales to marketing, enter **groupmod –n marketing sales**. An existing group can be deleted with the groupdel command. For example, to delete the marketing group, enter **groupdel marketing**.

Regardless of how you create or edit a group, the effect is the same. The updated group is listed in the /etc/group file. In the same manner that there are tools to create or modify a group, there are tools available for creating a user account.

Tools to Create User Accounts

All Linux distributions offer tools to help you create user accounts. Most distributions provide the useradd command for this purpose. The useradd command in Listing 3.11 creates a user account with the username kathy.

Listing 3.11: The Effect of the *useradd* Command

```
[root]# grep kathy /etc/passwd
[root]# grep kathy /etc/shadow
[root]# grep kathy /etc/group
[root]# ls -a /home/kathy
ls: /home/kathy: No such file or directory
[root]# useradd kathy
[root]# grep kathy /etc/passwd
kathy:x:501:501::/home/kathy:/bin/bash
[root]# grep kathy /etc/group
kathy:x:501:
[root]# ls -a /home/kathy
.  ..  .bash_logout  .bash_profile  .bashrc  Desktop  .gtkrc  .kde  .screenrc
[root]# grep kathy /etc/shadow
kathy:!!:11743:0:99999:7:::
```

The first four lines in Listing 3.11 show that there are no entries for the username kathy in the /etc/passwd, /etc/shadow or /etc/group file, and that the directory /home/kathy does not exist. Then the simple command useradd kathy is entered. The four commands after the useradd command show that much has changed. The simple useradd command creates the correct /etc/passwd, /etc/shadow, and /etc/group entries, it builds the new home directory (which by default is named /home/kathy), and it copies the /etc/skel files to the new directory. It does everything required for a successful login except create the login password. To define the login password, run passwd kathy, and enter the user's initial password.

This new account can be used as–is, yet it is not exactly what we want. First, the comment field of the /etc/passwd entry for kathy is empty. This field holds free form text that describes the user. We want it to contain the full name of the user. Second, kathy is a member of three groups. We want kathy in the groups users and popusers, as well as her primary group, which is kathy. The second grep of /etc/group in Listing 3.11 shows that she is in only one group—kathy. The usermod command can be used to change the settings for an existing user, as in this example:

Listing 3.12: Using the *usermod* Command

```
[root]# usermod -c "Kathleen McCafferty" -G users,popusers kathy
[root]# grep kathy /etc/passwd
kathy:x:501:501:Kathleen McCafferty:/home/kathy:/bin/bash
```

```
[root]# grep kathy /etc/group
users:x:100:kathy
popusers:x:45:kathy
kathy:x:501:
```

The usermod command in Listing 3.12 modifies the parameters for the account identified by username kathy. The string that follows the –c argument on the usermod command line is a text description for this user account. usermod places the string in the comment field of the /etc/ passwd entry for this account, as the first grep clearly shows. The –G argument defines supplement groups for the user account. In Listing 3.12, two supplemental groups, users and popusers, are defined. A grep of the /etc/group file shows that the usermod command added kathy to the user list for both of these groups.

Of course, instead of fixing an account with usermod, it is better to build it correctly from the start. The useradd command accepts the same arguments as the usermod command. Except for the fact that I wanted a nice example of the usermod command for this book, we could have created the kathy account with the following useradd command, and we would have had all of the settings we desired:

```
useradd -c "Kathleen McCafferty" -G users,popusers kathy
```

The useradd command accepts a wide range of options. The full syntax of the command is:

useradd [–c *comment*] [–d *home_dir*]

[–e *expire_date*] [–f *inactive_time*]

[–g *initial_group*] [–G *group*[,...]]

[–m [–k *skeleton_dir*] | –M]

[–p *passwd*] [–s *shell*] [–u *uid* [–o]]

[–n] [–r] *login*

The useradd command line arguments are

–c *comment* The comment field of the /etc/passwd entry for this user account.

–d *home_dir* The path to the home directory for this user account.

–e *expire_date* The date on which this account will be disabled. The date is entered as a four–digit year, a two–digit month and a two–digit day in the form *yyyy–mm–dd*. By default, accounts are not automatically disabled.

–f *inactive_time* The number of days after the password expires that the account will be permanently disabled. –1 turns off this feature, and –1 is the default.

–g *initial_group* The primary group for this user account.

–G *group*[,...]] A comma–separated list of supplement groups for this user account.

–m [–k *skeleton_dir*] | –M The –m option tells useradd to create the user's home directory, if one does not already exist. If –k is defined (and it can only be used if –m is also used), the files found in *skeleton_ dir* are copied to the newly created directory. If –k is not defined, the files found in /etc/skel are copied to the newly created directory. Alternatively, –M can be specified to tell useradd that it should not create a home directory for the new account.

–p *passwd* Defines the encrypted password for the new account. The value entered for *passwd* must already be encrypted. Additionally, it must be encrypted using the correct algorithm for your system. It is always easier to enter the password separately with the passwd command.

–s *shell* The pathname of the login shell to be used by this account.

–u *uid* [–o] Defines the numeric UID for this user account. The –o flag tells useradd not to reject the UID if it is a duplicate of another account's UID. Don't use duplicate UIDs. Plan your system so that all UIDs are unique, and so that each user has only one login account that clearly identifies that user. Doing anything else reduces system security.

–n Tells adduser that it should not create a group with the same name as the user account. As we saw when we created the user kathy, the Red Hat system also created a group named kathy. –n prevents this.

–r Indicates that the account should be created as a system account; that is, a non–login account that is not created for a user. The /etc/passwd example in Listing 3.9 showed several such accounts, for example, news, mail, and daemon.

login The username assigned to the account. useradd documentation calls this value the login, but it is easier and more accurate to think of it as the username.

Despite this long list of command–line options, all of the sample useradd commands shown in Listing 3.11 are relatively simple. That is because we used the command's default values. The useradd default values are defined in two files: /etc/default/useradd and /etc/login.defs.

Listing 3.13 shows the /etc/default/useradd file from our sample Red Hat system. The file contains keyword/value pairs. The keywords can be easily mapped to useradd command–line options. The values set in this file are the default values used for the command–line options. For example, HOME=/home tells useradd that if the –d option is not provided on the command line, it should use the login username to create a home directory for the user in the /home directory; and SHELL=/bin/bash tells useradd to use bash as the user's login shell if the –s option is not specified on the command line.

Listing 3.13: Contents of the */etc/default/useradd* File

```
[root]# cat /etc/default/useradd
# useradd defaults file
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
```

However, the /etc/default/useradd file is not the whole story. Look at the first keyword/value pair in Listing 3.13. It says GROUP=100, which tells useradd that if the –g option is not specified on the command line, it should use 100 as the default GID. Now, look back at Listing 3.12. There is no –g option on the useradd command line, yet the grep of the /etc/passwd file shows that the kathy account has been assigned the GID 501. Clearly, something else is at work here. That something else is the /etc/login.defs file. Listing 3.14 shows the /etc/login.defs file from our sample Red Hat system.

Listing 3.14: Contents of the */etc/login.defs* File

```
# *REQUIRED*
#   Directory where mailboxes reside, _or_ name of file, relative to the
#   home directory.  If you _do_ define both, MAIL_DIR takes precedence.
#   QMAIL_DIR is for Qmail
#
#QMAIL_DIR      Maildir
MAIL_DIR        /var/spool/mail
#MAIL_FILE      .mail

# Password aging controls:
#
#       PASS_MAX_DAYS   Maximum number of days a password may be used.
#       PASS_MIN_DAYS   Minimum number of days allowed between password changes.
#       PASS_MIN_LEN    Minimum acceptable password length.
#       PASS_WARN_AGE   Number of days warning given before a password expires.
#
PASS_MAX_DAYS   99999
PASS_MIN_DAYS   0
PASS_MIN_LEN    5
PASS_WARN_AGE   7

#
# Min/max values for automatic uid selection in useradd
#
UID_MIN                   500
UID_MAX                 60000

#
# Min/max values for automatic gid selection in groupadd
#
GID_MIN                   500
GID_MAX                 60000

#
# If defined, this command is run when removing a user.
# It should remove any at/cron/print jobs etc. owned by
# the user to be removed (passed as the first argument).
#
#USERDEL_CMD    /usr/sbin/userdel_local

#
# If useradd should create home directories for users by default
# On RH systems, we do. This option is ORed with the -m flag on
# useradd command line.
#
CREATE_HOME     yes
```

The /etc/login.defs file appears much larger and more complex than the /etc/default/useradd file. However, most of the login.defs file consists of comments. Every line that begins with a # is a comment. A liberal dose of comments is provided to explain the purpose of the various parameters defined in this file. Only ten parameters are really defined in Listing 3.14.

MAIL_DIR /var/spool/mail The MAIL_DIR parameter defines what type of mail directory should be created for the new user, and where it should be created. Three options are available: a standard mail directory (MAIL_DIR), a Qmail directory (QMAIL_DIR), or a mail file in the user's home directory (MAIL_FILE). The Red Hat configuration in Listing 3.14 creates a standard mail directory in /var/spool/mail. After running the useradd command shown in Listing 3.12, an ls of /var/spool/mail would show a new mail directory named kathy.

PASS_MAX_DAYS 99999

PASS_MIN_DAYS 0

PASS_MIN_LEN 5

PASS_WARN_AGE 7 These four parameters define the values used for creating passwords. Three of the parameters (PASS_MAX_DAYS, PASS_MIN_DAYS and PASS_ WARN_AGE) are used for password aging. Password aging is described in Chapter 12 as part of the discussion of the /etc/shadow file. The grep of the /etc/shadow file in Listing 3.11 shows the three values (0, 99999, and 7) in the entry for the newly created kathy account. The fourth parameter, PASS_MIN_LEN, defines the number of characters that a password must exceed to be acceptable. In the Red Hat configuration, this is set to 5, meaning that a password must be at least six characters long to be acceptable.

UID_MIN 500

UID_MAX 60000 These two parameters define the values used to automatically assign a UID. UIDs will be automatically assigned in sequential order, starting from 500. The highest UID that will be assigned is 60000.

GID_MIN 500

GID_MAX 60000 These two parameters define the values used to automatically assign a GID. GIDs will be automatically assigned in sequential order starting from 500. The highest GID that will be assigned is 60000. These are the parameters that overrode the GROUP=100 setting in the /etc/default/useradd file. The kathy account was assigned GID 501 because it was the second account created on this system. The first account was assigned 500, and the kathy account was assigned 501.

CREATE_HOME yes This parameter tells useradd whether or not it should create a home directory when creating a new user account. If this parameter is set to yes, useradd creates the home directory, whether or not the –m option is used on the command line. If this parameter is set to no, useradd creates the home directory only if –m is specified on the command line. Regardless of how this parameter is set, using –M on the useradd command line prevents the home directory from being created.

Although the 10 parameters listed previously were set in the sample /etc/login.defs file shown in Listing 3.14, there is one parameter that was not set. The USERDEL_CMD parameter is commented out of the default Red Hat login.defs file. USERDEL_CMD defines the path to the userdel command if a non–standard userdel command is used. Because this system uses the standard userdel command, it has no need to define a special path with the USERDEL_CMD parameter.

Use userdel to remove an account that is unneeded or unused. To remove the kathy account, simply enter **userdel kathy**, as shown in Listing 3.15.

Listing 3.15: The *userdel* Command

```
[root]# userdel kathy
[root]# grep kathy /etc/passwd
[root]# grep kathy /etc/shadow
[root]# grep kathy /etc/group
[root]# ls -a /home/kathy
.  ..  .bash_logout  .bash_profile  .bashrc  Desktop  .gtkrc  .kde  .screenrc
[root]# ls /var/spool/mail
alana  kathy  nfsnobody  root
```

The userdel command removes the kathy account from all of the administrative files, as shown by the grep searches of the passwd, shadow, and group files. It does not, however, remove any of the user's files, as shown by the two ls commands. To force userdel to remove all of the user's files, run it with the –r option (for example, **userdel –r kathy)**.

The useradd, usermod, and userdel tools, simplify user account maintenance by automatically performing the tasks described in the section "The Steps to Creating a User Account." User accounts are a fundamental component of a complete network server. The user account identifies and authenticates the user at login, and provides the UID and GID used for filesystem and process security—both locally and remotely through the network.

Additional FTP Configuration

One final topic before leaving the subject of network login services: the FTP service. Like other services, to access the FTP server, a user must provide a username and a password. The sample user kathy could ftp to the local system, and log in. The FTP server would set her default directory to/home/kathy, and she would be able to download and upload files to and from the system based on her normal file read and write permissions.

In addition to its standard service, ftp provides anonymous FTP, which allows anyone to log in to the FTP server with the username anonymous and any password. Traditionally, the password used is your e–mail address. The purpose of anonymous FTP is, of course, to make certain files publicly available. Many of the great Linux files available from the Internet are available through anonymous FTP.

Anonymous FTP is a great service, but it can present a security problem—and a big headache— if it is set up incorrectly. Several steps are involved in doing it right. The steps to create an anonymous FTP server are as follows:

1. Add the user ftp to the /etc/passwd file.

2. Create an ftp home directory owned by user ftp that cannot be written to by anyone.
3. Create a bin directory under the ftp home directory that is owned by root, and that cannot be written to by anyone. The programs needed by FTP should be placed in this directory.
4. Create an etc directory in the ftp home directory that is owned by root, and that cannot be written to by anyone. Create special passwd and group files in this directory, and change the mode of both files to 444 (read–only).
5. Create a pub directory in the ftp home directory that is owned by root and is only writable by root; that is, mode 644. Don't allow remote users to store files on your server unless it is absolutely necessary and your system is on a private, non–connected network. If you must allow users to store files on the server, change the ownership of this directory to ftp and the mode to 666 (read and write). This should be the only directory in which anonymous FTP users can store files.
6. For systems, such as Linux, that use dynamic linking, create a lib directory in the ftp home directory that contains the runtime loader and the library modules needed by FTP.

On Linux, setting up anonymous FTP is simple because the steps described previously have already been done for you. Most Linux systems come with anonymous FTP preconfigured and installed. Simply select the anonymous FTP component during the initial installation, or add it later using a package manager. Figure 3.1 shows the results of a Gnome RPM query for the anonymous FTP package on a Red Hat 7.2 system.

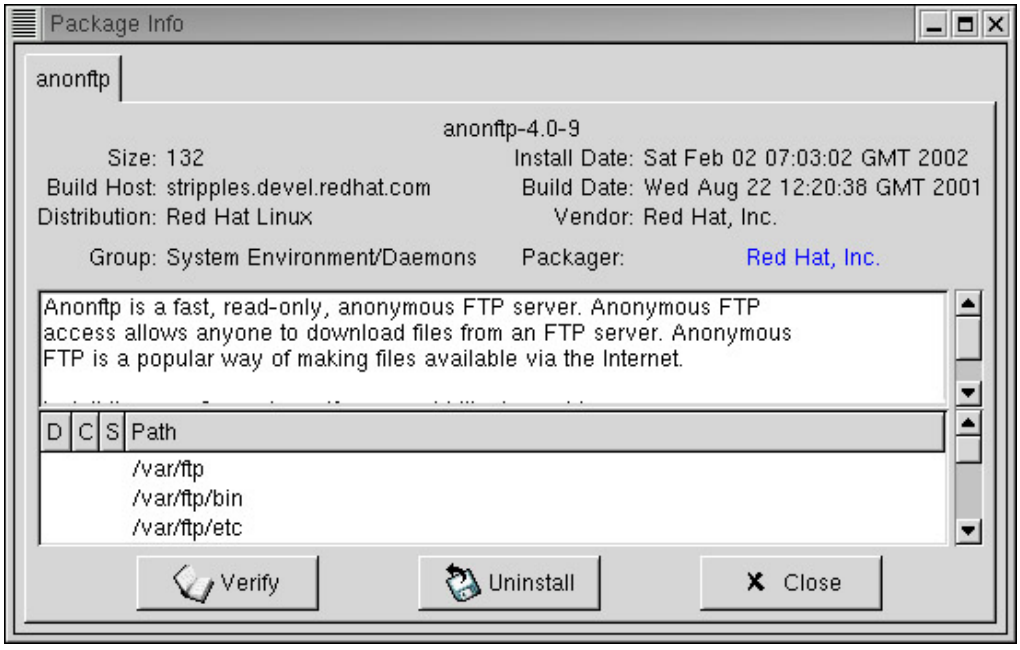


Figure 3.1: The anonymous FTP RPM

Figure 3.1 shows a Red Hat system in which the anonymous FTP package has already been installed. The effects of the installation are visible on the system. Look in the /etc/passwd file; you'll notice that the user account ftp is already there. You'll also find the anonymous FTP home directory, which is /var/ftp on a Red Hat 7.2 system. Finally, test the system with the command ftp localhost, and you should be able to log in as anonymous.

Properly set up, anonymous FTP is less of a security risk than regular FTP. If you don't want to offer an FTP server at all, comment the ftp entry out of the inetd.conf file, or disable it in the xinetd configuration. If you specifically don't want anonymous FTP, don't install it in the first place, or comment the ftp entry out of the /etc/passwd file if it is already installed.

Basic FTP and anonymous FTP are the only FTP services offered on most Linux systems. Basic

service is configured by enabling the service through xinetd or inetd, and by creating user accounts. Anonymous FTP is configured by installing the anonymous FTP package. For many Linux systems, this is all there is to FTP configuration. However, Linux systems that use Washington University FTP (WU–FTPD) have additional configuration options.

The *ftpaccess* File

WU–FTPD has an optional configuration file named /etc/ftpaccess. This file is read if the FTP daemon is run with the –a command line option. In the discussion of Listing 3.6, we saw that Red Hat does run the FTP daemon with the –a option, which means that Red Hat uses the ftpaccess file. The active entries in the Red Hat 7.2 ftpaccess file are shown in Listing 3.16.

Listing 3.16: Excerpts of the Red Hat *ftpaccess* File

```
# Don't allow system accounts to log in over ftp
deny-uid %-99 %65534-
deny-gid %-99 %65534-
allow-uid ftp
allow-gid ftp
# To chroot a user, modify the line below or create
# the ftpchroot group and add the user to it.
guestgroup ftpchroot
# User classes...
class   all   real,guest,anonymous  *
# Set this to your email address
email root@localhost
# Allow 5 mistyped passwords
loginfails 5
# Notify the users of README files at login and cwd
readme  README*    login
readme  README*    cwd=*
# Messages displayed to the user
message /welcome.msg            login
message .message                cwd=*
# Allow on-the-fly compression and tarring
compress        yes             all
tar             yes             all
# Prevent anonymous users (and partially guest users)
# from executing dangerous commands
chmod           no              guest,anonymous
delete          no              anonymous
overwrite       no              anonymous
rename          no              anonymous
# Turn on logging to /var/log/xferlog
log transfers anonymous,guest,real inbound,outbound
# If /etc/shutmsg exists, don't allow logins
shutdown /etc/shutmsg
# Use user's email address as anonymous password
passwd-check rfc822 warn
```

Blank lines, inactive lines, and most of the comments have been removed from the ftpaccess file to create a listing that is more suitable for a book. However, all of the active commands used in the Red Hat configuration are shown in Listing 3.16.

The deny–uid and deny–gid commands define ranges of UIDs that are not allowed to log in to the FTP server. In Listing 3.16, UIDs and GIDs that are less than 99 (%–99) or greater than 65534 (65534–%) are not allowed to log in. This blocks all of the UIDs and GIDs that are normally used for

system accounts. However, on the Red Hat system, this also blocks the ftp user account that is used for anonymous FTP because that account is assigned UID 14 and GID 50. The allow−uid and allow−gid commands define exceptions to the rules defined by the deny−uid and deny−gid commands. Therefore, all of the system accounts except ftp are prevented from logging into this FTP server.

To understand the next command, you need to understand that WU−FTPD offers three types of service:

- Real FTP, in which a user logs in with a standard username and password, and is granted access to files based on the UID and GID associated with that user's account.
- Anonymous FTP, in which the user logs in as an anonymous guest, and is limited to those files stored in the anonymous FTP home directory.
- Guest FTP, in which the user logs in with a standard username and password, and is limited to those files found in the user's home directory.

The guestgroup command defines the users who are limited to guest FTP service. The value that follows the command guestgroup must be a valid group name from the /etc/group file. Every user account listed as a member of that group is limited to the guest FTP service. As the comment indicates, ftpchroot does not exist as a group unless you create it. The guestgroup command in the Red Hat configuration is only an example. It has no real effect.

If you do decide to limit a user to guest FTP, you must create the same file structure in the user's home directory as was created in the anonymous FTP home directory. See the steps outlined previously for creating an anonymous FTP service, and duplicate all of those steps in the home directory of each user that you limit to guest FTP.

The class command maps the source address of the FTP connection to a user "class." The format of the command is

class *name type address*

where class is the keyword. *name* is the arbitrary name we are assigning to the class. *type* is the type of FTP service being used, which is either anonymous, guest, or real. And *address* is the source address of the connection, written as either a domain name or an IP address.

In Listing 3.16, the class command assigns the name all to anonymous, guest, and real connections from all sources. The * is a wildcard character that matches anything. Therefore a * by itself matches all addresses, whereas *.foobirds.org matches all hosts in the foobirds.org domain. After the class all is defined, it can be used in other configuration commands.

The email command defines the email address of the FTP server system administrator. Change this to a valid e−mail address.

Use loginfails to set a limit on the number of times a user can enter the wrong password before the session is terminated. Three incorrect passwords is a common value. In Listing 3.16, the value is set to 5.

The readme commands notify the user that a README file exists when the user logs in or changes directories. The message is issued only if the directory to which the user changes contains a file with a name that matches the filename on the readme command line. In Listing 3.16, the filename in both commands is README*, which matches any filename that begins with the string README.

The message commands perform a similar task. The message commands point to files that contain the welcome messages that are displayed when the user logs into the system and when the user changes directories.

The compress and tar commands specify whether or not on−the−fly compressing and tarring are allowed, and who is allowed to use these services. In Listing 3.16, both services are allowed, and they are allowed for all types of users. Note that "all" is the class defined earlier in this ftpaccess file.

In the same way that services can be allowed, specific FTP commands can be disallowed. Listing 3.16 forbids anonymous FTP users from changing the permission of a file (chmod), deleting files (delete), overwriting files (overwrite), and renaming files (rename); also it prevents users of the guest FTP service from changing file permissions.

The log command specifies what FTP should log and when it should be logged. In the example, FTP will log file transfer statistics for users of the anonymous, guest, and real FTP servers, for both uploads and downloads. The log command can also be used to log the commands invoked by the users and any violations of security rules.

The shutdown command points to the file that directs the FTP server to cease operations. Based on the shutdown command in Listing 3.16, this server will shut down if instructed to do so by the file /etc/shutmsg. The file contains the year, month, hour, and minute the server should shut down, along with an offset from the shutdown time that the server should stop accepting connections, and a second offset for when it should break connections that were previously established. Additionally, the file can contain a text message to be displayed prior to the shutdown.

The last command in this sample ftpaccess file is passwd−check. It tells FTP to warn anonymous users who do not enter an email address as their password, but to accept the user's login anyway. To prevent users from logging in who do not enter an e−mail address, the keyword warn at the end of the sample passwd−check command must be changed to enforce.

The Red Hat ftpaccess file shown in Listing 3.16 is a typical WU−FTPD configuration. There are, however, additional features available for WU−FTPD. To find out more, see the ftpaccess main page and the HOWTO files in /usr/share/doc/wu−ftpd−2.6.1.

In Sum

Telnet service permits a user to connect to the server and run a program there. FTP allows users to move files into and out of the server. These simple, basic services are traditional components of all TCP/IP networks. The Internet service daemon (inetd) or the Extended Internet service daemon (xinetd) starts these and many other network services. The system administrator must configure inetd.conf or xinetd.conf to start the services that will be offered by his network server.

Telnet, FTP, and many other network services need to identify the user to grant access and control file permission. Therefore, the users of many network services must have valid user accounts on the server. Linux provides an array of tools that the system administrator can use to maintain user accounts.

The extensive list of network services started by inetd or xinetd is not the whole story. Some of the most important network services are started independently of inetd and xinetd. The next four chapters discuss four of these important services: Domain Name System, sendmail, Web service, and routing. The discussion of these Internet services begins in Chapter 4, with the configuration of

the Domain Name System (DNS).

Chapter 4: Linux Name Services

Overview

One of the most fundamental services on a TCP/IP network is *name service*. It is the service that translates hostnames into IP addresses. In Chapter 3, "Login Services," we configured telnet and ftp. Without name service, a user connecting to crow enters

```
telnet 172.16.5.5
```

With name service, that same command is

```
telnet crow
```

The result is the same. In either case, the user connects to the host at address 172.16.5.5. But most users prefer hostnames because they are easier to remember and easier to use. This is particularly true in the global Internet. It is possible to guess that http://www.sybex.com/ is a valid name, but there is no intuitive way to guess the address 206.100.29.83.

Linux systems use two techniques to convert hostnames to addresses: the host table and the Domain Name System (DNS). The /etc/hosts file is a table that maps names to addresses. It is a simple text file that is searched sequentially to match hostnames to IP addresses. The Domain Name System is a hierarchical, distributed database system with thousands of servers across the Internet, handling name and address queries. DNS is far more important than the host table for the operation of the Internet, but both services play a role. This chapter's discussion of name services begins with a quick look at the host table.

The *hosts* File

Each entry in the /etc/hosts file contains an IP address and the names associated with that address. For example, the host table on crow might contain the following entries:

Listing 4.1: A Sample Host Table

```
$ cat /etc/hosts
127.0.0.1               localhost localhost.localdomain
172.16.5.5              crow.foobirds.org crow
172.16.5.1              wren.foobirds.org wren
172.16.5.2              robin.foobirds.org robin
172.16.5.4              hawk.foobirds.org hawk
```

The first entry in this table assigns the name localhost to the address 127.0.0.1. (Every computer running TCP/IP assigns the loopback address to the hostname localhost.) Network 127 is a special network address reserved for the loopback network, and host 127.0.0.1 is the loopback address reserved for the local host. The *loopback address* is a special convention that permits the local computer to address itself in exactly the same way that it addresses remote computers. This simplifies software because the same code can be used to address any system, and because the address is assigned to a software loopback interface (lo), no traffic is sent to the physical network.

The second entry in the table is the entry for crow itself. The entry maps the address 172.16.5.5 to the name crow.foobirds.org and to the alias crow. Alias hostnames are primarily used to provide for shorter names, as in the example, but they are also used to provide generic names such as mailhost, news, and www. Every networked computer with a permanent address has its own hostname and address in its host table.

Every Linux system has a host table with the two entries just discussed, and some, such as the system in Listing 4.1, have a few additional entries for other key systems on the local network. This small table provides a backup for those times when DNS might not be running, such as during the boot process.

Understanding DNS

The limitations of the host table become obvious when it is used for a large number of hosts. The host table requires every computer to have a local copy of the table, and each copy must be complete because only computers listed in the local host table can be accessed by name.

Consider today's Internet: It has millions of hosts. A host table with millions of entries is very inefficient to search and, more important, is impossible to maintain. Hosts are added to and deleted from the Internet every second. No system could maintain such a large and changeable table and distribute it to every computer on the network.

DNS solves these problems by eliminating the need for an all–inclusive, centrally maintained table by replacing it with a distributed, hierarchical database system. The current DNS database has millions of host entries, distributed among tens of thousands of servers. Distributing the database in this way reduces the size of the database handled by any individual server, which in turn reduces the task of maintaining any individual piece of the database.

Additionally, DNS uses local caching to migrate information close to those who need it, without sending unnecessary information. A caching server starts with just the information it needs to locate the root of the hierarchical database. It then saves all of the answers to user queries that it receives and all of the supporting information learned in gaining those answers. In this way, it builds up an internal database of the information it needs to serve its users.

The DNS Hierarchy

The DNS hierarchy can be compared to the hierarchy of the Linux filesystem. Hostnames in individual domains parallel filenames in individual directories, and, like the root directory of the filesystem, DNS has a root domain.

In both the filesystem and the DNS system, the names of objects reveal the rooted hierarchical structure. Filenames move from the most general, the root (/), to the most specific, the individual file. Domain names start with the most specific, the host, and move to the most general, the root (.). A domain name that starts with a host and goes all the way to the root is called a *fully qualified domain name* (FQDN). For example, wren.foobirds.org is the FQDN of one of the systems on our imaginary network.

The top–level domains (TLDs)—such as org, edu, jp, and com—are serviced by the root servers. The second–level domain, foobirds in the example, is the domain that has been officially assigned to our imaginary organization. When you're officially assigned a domain by your parent domain, a pointer is placed in the parent domain that points to your server as the server responsible for your

domain. It is this delegation of authority that makes your domain part of the overall domain system. How to delegate authority for subdomains is covered later in this chapter.

Note This book assumes that you already have an official domain name and IP address. If you don't, and you need information on how to obtain a domain name or IP address, see *TCP/IP Network Administration*, Third Edition, by Craig Hunt (O'Reilly, 2002).

The analogy to the filesystem goes beyond just the structure of names. Files are found by following a path from the root directory through subordinate directories to the target directory. DNS information is located in a similar manner. Linux learns the location of the root filesystem during the boot process from the grub.conf file or the lilo.conf file. Similarly, your DNS server locates the root servers during startup by reading a file, called the *hints file*, which contains the names and addresses of the root servers. (You will create that file later in this chapter.) Via queries, the server can find any host in the domain system by starting at the root and following pointers through the domains until it reaches the target domain.

Answering Queries

To answer a query for DNS information, the local name server must either have the answer to the query or know which name server does. No single system can have complete knowledge of all of the names in the Internet; servers know about their local domains and build up knowledge about other domains one query at a time.

Here's how it works. Assume you want the address of http://www.sybex.com/. In effect, you are asking for the address record for www from the sybex.com database. A query for that address record comes to the local name server. If the server knows the address of http://www.sybex.com/, it answers the query directly. If it doesn't know the answer, but it knows which server handles sybex.com, it queries that server. If it has no information at all, it queries a root server.

The root server does not directly answer the address query. Instead, it points the local server to a server that can answer queries for the sybex.com domain. It does this by sending the local server a name server record that tells it the name of the server for the sybex.com domain and an address record that tells it the address of that server. The local server then queries the sybex.com domain server and receives the address for http://www.sybex.com/.

In this way, the local server learns the address of the host as well as the name and address of the servers for the domain. It caches these answers and will use them to directly answer queries about the sybex.com domain without again bothering the root servers.

The BIND Software

On most Linux systems, DNS is implemented with the Berkeley Internet Name Domain (BIND) software. Two versions of BIND are currently in widespread use:

- BIND 8 has been around for years, and is found in many releases of Linux.
- BIND 9 is the most recent version of BIND, and is found on some new Linux distributions, such as Red Hat 7.2. This chapter focuses on BIND 9.

Note If you are running BIND 8, the information covered here is still applicable because BIND 8 and BIND 9 are very similar. Any differences are noted in the text.

BIND DNS is a client/server system. The client is called the *resolver*, and it forms the queries and sends them to the name server. Every computer on your network runs a resolver. Many systems only run a resolver.

Traditionally, the BIND resolver is not implemented as a distinct process. It is a library of software routines, called the *resolver code*, which is linked into any program that requires name service. Most Linux systems use the traditional resolver implementation, which is called a *stub resolver*. Because it is the most widely used, the stub resolver gets most of the coverage in this chapter. However, BIND 9 also offers the Lightweight Resolver library and the Lightweight Resolver daemon (lwresd) as an alternative to the traditional resolver. Systems running BIND 9 do not have to use lwresd, and Red Hat 7.2 does not use it. However, we do cover lwresd later in the chapter so that you know when and how it is used.

The server side of BIND answers the queries that come from the resolver. The name server daemon is a distinct process called named. The configuration of named is much more complex than the configuration of the resolver, but there is no need to run named on every computer. (See "The named Configuration File" section later in this chapter for more on named and the named.conf file.)

Because all of the computers on your network—whether they are clients or servers—run the resolver, begin your DNS configuration by configuring the resolver.

Configuring the Resolver

The Linux resolver is configured by two types of files. One type tells the resolver which name services to use and in what order to use them. That topic is discussed at the end of this chapter. The other configuration file, /etc/resolv.conf, configures the resolver for its interaction with the Domain Name System. Every time a process that uses the resolver starts, it reads the resolv.conf file, and caches the configuration for the life of the process. If the /etc/resolv.conf file is not found, a default configuration is used.

The default resolver configuration uses the hostname command to define the local domain. Everything after the first dot in the value returned by the hostname command is used as the default domain. For example, if hostname returns wren.foobirds.org, the default value for the local domain is foobirds.org.

The default configuration uses the local system as the name server. This means that you must run named if you don't create a resolv.conf file. Generally, I think you should create a resolv.conf file, even if you do run named on the local host. There are three reasons for this. First, the resolv.conf file provides a means of documenting the configuration. Anyone can read the file and see the configuration you selected for the resolver. Second, the default values that work with one version of BIND may change with a future release. If you explicitly set the values you want, you don't have to worry about how the default values change. And third, even if you are using the local machine as a name server, the resolv.conf file allows you to define backup servers to increase robustness.

The Resolver Configuration Commands

The BIND 9 software delivered with Red Hat 7.2 uses the same resolver configuration file as the BIND 8 software used on many Linux systems. The commands it contains are identical to those found in a BIND 8 resolver configuration. resolv.conf is a text file that can contain the following commands:

nameserver *address* The nameserver command defines the IP address of a name

server the resolver should use. Up to three nameserver commands can be included in the configuration. The servers are queried in the order that they appear in the file until an answer is received or the resolver times out. (See the "Resolver Timeouts" sidebar for information about these timeouts.) Normally, the first name server answers the query. The only time the second server is queried is if the first server is down or unreachable. The third server is queried only if both the first and second servers are down or unreachable. If no nameserver entry is found in the resolv.conf file, the name server running on the local host is used as the default.

domain *domainname* The domain command defines the local domain. The local domain is used to expand the hostname in a query before it is sent to the name server. If the domain command is not used, the values defined in the search command are used. If neither command is found in the resolv.conf file, the value derived from the hostname command is used. No matter how the local domain value is set, it can be overridden for an individual user by setting the environment variable LOCALDOMAIN.

search *searchlist* The search command defines a list of domains that are used to expand a hostname before it is sent to the name server. *searchlist* contains up to six domain names, separated by whitespace. Each domain specified in the search list is searched in order until the query is answered. Unlike the domain command, which creates a default search list containing only the local domain, the search command creates an explicit search list containing every domain specified in *searchlist*.

options *option* The options command modifies the standard behavior of the resolver. There are several options available for BIND 8 and BIND 9:

> **debug** Turns on debugging. When the debug option is set, the resolver prints debugging messages to standard output. These messages are very informative for debugging resolver or server problems, but in reality, this option is of marginal value. Turning on debugging in the basic resolver configuration produces too much output, and produces it at inappropriate times. Use the debugging tools described in Chapter 13, "Troubleshooting." The DNS test tools described there allow you to turn on resolver debugging when you're ready to run the test so that you get the additional output at a time that you can actually use it.
>
> **ndots:*n*** Defines the number of dots that indicate when the resolver should append values from the search list to the hostname before sending the query to the name server. By default, the resolver will not modify a hostname if it contains one dot. As a result, the hostname crow will be extended with a value from the search list before being sent to the name server, but the hostname sooty.terns will not. Use the ndots option to modify this behavior. For example: ndots:2.
>
> This option tells the resolver to use the search list on any hostname that contains less than two dots. With this setting, both crow and sooty.terns are extended with a value from the search list before being sent to the name server for the first time.
>
> The ndots option changes how the resolver handles queries, but it

only really changes the order in which things are done. The unmodified hostname is either the first or the last query sent to the name server. If ndots is set to 1 (the default setting), sooty.terns is sent to the server without modification. When the server fails to resolve that name, the resolver issues additional queries with the search list values until it gets an answer or the list is exhausted. If ndots is set to 2, sooty.terns is modified with the first value in the search list before it is sent to the name server. If the server fails to resolve the name, the resolver tries every value in the search list and then sends the unmodified hostname to the name server. In either case, exactly the same queries are made. Only the order of the queries is changed.

About the only time that ndots is required is if some component of your domain could be confused with a top–level domain, and you have users who consistently truncate hostnames at that domain. In that rare case, the queries would first be sent to the root servers for resolution in the top–level domain before eventually getting back to your local server. It is very bad form to bother the root servers over nothing. Use ndots to force the resolver to extend the troublesome hostnames with your local domain name so that they will be resolved before ever reaching the root servers.

timeout:*n* Sets the initial query timeout for the resolver. By default, the timeout is five seconds for the first query to every server. Subsequent queries time out based on a formula that uses this initial value and the number of servers (see the "Resolver Timeouts" sidebar for a detailed explanation). Change this value only if you know for certain that your name server generally takes longer than five seconds to respond. In that rare case, increasing this value reduces the number of duplicate queries.

attempts:*n* Defines the number of times the resolver will retry a query. The default value is 2, which means the resolver will retry a query two times with every server in its server list before returning an error to the application. The attempt value might need to be increased if you have a poor network connection that frequently loses queries, such as the connection to a remote office in a developing country or at the end of a narrow–band satellite link. In most cases, this value does not need to be changed.

rotate Turns on round–robin selection of name servers. Normally, the resolver sends the query to the first server in the name server list, and only sends the query to another server if the first server does not respond. Traditionally, the second and third name servers were defined to provide backup name service. They were not intended to provide load–sharing. The rotate option configures the resolver to share the name server workload evenly among all of the servers. Here's how it works: Assume that the resolv.conf file has the following nameserver entries:

```
nameserver 172.16.5.1
nameserver 172.16.5.3
```

```
nameserver 172.16.55.1
```

Furthermore, assume that FTP has asked the resolver for the address of crow, Telnet has asked for the address of kestrel, and Apache has asked for the address of grackel. Without the rotate option set, all three address queries are sent to the name server at 172.16.5.1. With the rotate option set, the query for crow is sent to the server at 172.16.5.1, the query for kestrel is sent to the server at 172.16.5.3, and the query for grackel is sent to 172.16.55.1. The resolver starts at the top of the server list, sends a query to each server in the list, and then starts at the top again.

Implementing load–sharing at the resolver level makes sense only if you have a large number of resolvers involved, and you have more than one robust server. For example, assume that you have a large enterprise with 50,000 clients, and that the resolvers of all of those clients are configured in exactly the same way. All 50,000 would send all of their queries to the first server in the list of name servers. The rotate option would spread the work evenly among all of the central servers.

This, however, is not usually the case. Most resolver configurations list a local name server (such as the name server on the local subnet) first, and list other servers only as backup servers. With this model, there are as many different resolver configurations as there are subnets, and no server is targeted by more than the number of clients on a single subnet. In this, the average case, setting the rotate option is unnecessary and even undesirable because the topology of the network already balances the load.

no–check–names Disables the checking of domain names for compliance with RFC 952, "DOD Internet Host Table Specification." By default, domain names that contain an underscore (_), non–ASCII characters, or ASCII control characters are considered to be in error. Use this option to work with hostnames that contain an underscore.

Philosophically, I'm not crazy about checking for bad names during the query process. Checking names at this point does not seem to be in the spirit of the old interoperability adage, "Be conservative in what you send and liberal in what you accept." Personally, I prefer to control compliance with the RFCs at the source. The sources of incorrect domain names are the zone files that contain those names. I find it much better to use the name server features that check for incorrect names when the zone file is loaded than to check the names during a resolver query process. Therefore, I use this option to disable checking in the resolver.

inet6 Causes the resolver to query for IPv6 addresses. The version of the Internet Protocol (IP) used in today's Internet is IPv4. IPv4 uses the same 32–bit addresses used in this book. IPv6 uses different 128–bit addresses. IPv6 is a future protocol toward which networks are evolving. Use this option only if you connect to an experimental IPv6 network. This option would not be used in an average business

environment.

sortlist *addresslist* The sortlist command defines a list of network addresses that are preferred over other addresses. The *addresslist* is a list of address and mask pairs (for example, 172.16.5.0/255.255.255.0). To be of any use, it requires that the remote host have more than one address assigned to a single name, that the network path to one of those addresses be clearly superior to the others, and that you know exactly which path is superior. By default, address preference is set by the server, and addresses are used by the resolver in the order in which they are received.

A sortlist command that prefers the local subnet address above all others is the only sortlist that is commonly used. Many administrators use such a sortlist command as a matter of course. But trying to use the sortlist command to sort addresses from remote networks can do more harm than good unless you have a very clear understanding of exactly how the networks are configured and how they talk to each other. Therefore sortlist is rarely used to sort remote addresses.

Resolver Timeouts

The standard timeout for the first query is set to five seconds, and the resolver can use up to three name servers. The resolver sends the first query to the first server in the server list with a timeout of five seconds. If the first server fails to respond, the query is sent to the second server in the list with a timeout of five seconds. If the second server fails to respond, the query is sent to the third server in the list with a time–out of five seconds. Each server in the list is given a chance to respond, and each server is given the full five–second timeout. This first round of queries can take as long as 15 seconds.

If no server responds to the first round of queries, the previous timeout is doubled and divided by the number of servers to determine the new timeout value. The query is then sent again. By default, the resolver retries two times, although this can be changed with the retry command. Assuming the defaults, the resolver sends an initial query and two retries for a total of three attempts, which gives the following timeouts:

Timeouts with one server The first query has a timeout of five seconds, the second of 10 seconds, and the third of 20 seconds. Given this, the resolver waits up to 35 seconds before abandoning the query when one server is defined.

Timeouts with two servers The first round of queries has a timeout of five seconds for each server, the second round is also five seconds, and the third round is 10 seconds. This gives a total timeout value of 20 seconds for each server, which means the resolver waits up to 40 seconds before abandoning a query when two servers are defined.

Timeouts with three servers The first round of queries has a timeout of five seconds for each server, the second round is three seconds, and the third round is six seconds. (Dividing 10 or 20 by 3 does not yield a whole number—the time–out values 3 and 6 are truncated, whole number values.) This gives a total timeout value of 14 seconds for each server, which means the resolver waits up to 42 seconds before abandoning a query when three servers are defined.

Without this formula, which reduces the timeout based on the number servers, two retries with three servers would take up to 105 seconds to time out if the resolver used the same 5–, 10–, and 20–second timeout values used for one server. Even the most patient user would become exasperated! Additionally, when multiple name servers are used, it is not necessary to give each of them as much time to resolve the query. It is highly unlikely that they will all be down at the same time. A query timeout when three servers are configured probably indicates that there is something wrong with your local network, not with all three remote servers. Because of this timeout formula, the added reliability of three servers only costs, at most, seven seconds.

A Sample *resolv.conf* File

Assume you're configuring a Linux workstation named mute.swans.foobirds.org (172.16.12.3) that does not run its own name server. Listing 4.2 shows a reasonable resolv.conf file.

Listing 4.2: A Sample */etc/resolv.conf* File

```
$ cat /etc/resolv.conf
search swans.foobirds.org foobirds.org
nameserver 172.16.12.1
nameserver 172.16.5.1
```

The configuration has two nameserver entries. The address of the first name server is 172.16.12.1. It's on the same subnet as mute. The other name server (172.16.5.1) is the main server for the foobirds.org domain. For efficiency's sake, send queries to the server on the local subnet. For backup purposes, send queries to the main domain server when the local server is down.

The search command tells the resolver to expand hostnames, first with the local subdomain swans.foobirds.org and then with the parent of that domain foobirds.org. This explicit list gives the workstation's users the behavior they have come to expect. In earlier versions of BIND, the default was to search the local domain and its parents. The default in BIND 8 and BIND 9 is to search only the default domain. This explicit search list emulates the old behavior.

The domain command and the search command are mutually exclusive. Whichever command appears last in the resolv.conf file is the one that defines the search list. To have more than one domain in the search list, use the search command. The default value derived from the hostname command, the value entered by the domain command, and the value assigned to the LOCALDOMAIN environment variable defines just one domain—the local domain. The local domain then becomes the only value in the search list. The search command is the preferred method for defining the search list.

The Lightweight Resolver

BIND 9 introduces a new, lightweight resolver library. The new library can be linked into any application, but it was designed for applications that need to use IPv6 addresses. Support for IPv6 has increased the complexity of the resolver to the point that it is difficult to implement as a traditional stub resolver. For this reason, the lightweight resolver splits the resolver into a library used by the applications and a separate resolver daemon that handles the bulk of the resolver process. The library routines send queries to UDP port 921 on the local host using the lightweight resolver protocol. The resolver daemon takes the query, and resolves it using standard DNS protocols.

The resolver daemon is lwresd. It is essentially a caching–only name server that recursively resolves queries for the lightweight resolver library. lwresd does not require the same level of configuration as a caching–only server because some default values, such as the list of root servers, are compiled into lwresd. Instead, lwresd uses the same configuration commands as the stub resolver, and it reads its configuration from resolv.conf. However, lwresd interprets the nameserver commands in a slightly different manner. If nameserver commands are defined in resolv.conf, lwresd treats the servers listed there as forwarders. It attempts to forward all queries to those servers for resolution.

lwresd is started by the lwresd command. Red Hat does not include a boot script for lwresd. If you need to run lwresd to support IPv6 on your network, add a lwresd command to the rc.local startup script. (See Chapter 1, "The Boot Process," for information on rc.local.) The lwresd command uses the following syntax:

```
lwresd [-C config-file] [-d debuglevel] [-f -g -s] [-i pid-file]
       [-n #threads] [-P listen-port] [-p port] [-t directory]
       [-u user-id] [-v]
```

The lwresd command has several arguments, most of which you will never use. They are as follows:

–C *config–file* Defines the pathname of the configuration file if /etc/resolv.conf is not used. It is always best to use the standard configuration file to simplify troubleshooting.

–d *debuglevel* Enables debug tracing. This is the same debugging output used with named.

–f Runs lwresd in the foreground instead of running it in the background as a standard daemon. Generally, this is used only for debugging.

–g Runs lwresd in the foreground, and logs everything through stderr. Again, this is used only for debugging.

–s Writes memory usage statistics to stdout. This is of interest only to the BIND developers.

–i *pid–file* Defines the pathname of the file in which lwresd writes its process ID. The default is /var/run/lwresd.pid. Changing default pathnames gains nothing and complicates troubleshooting.

–n *#threads* Specifies the number of threads that should be created by lwresd. By default, lwresd creates one thread for each CPU.

–P *listen–port* Specifies the port that should be used for queries from the lightweight resolver library. The standard port is 921. Changing the port complicates troubleshooting.

–p *port* Specifies the port on which DNS queries should be sent to the name servers. The standard port is port 53. This is changed only for special testing with a server that uses a non–standard port.

–t *directory* Defines a chroot directory to run lwresd chroot. If –t is used, lwresd will chroot to *directory* after reading /etc/resolv. This is used to limit the access that lwresd has to the filesystem in case the lwresd process is compromised by an intruder.

–u *user–id* Defines the user ID under which the lwresd process will run. lwresd starts under the root user ID, and changes to *user–id* after completing privileged operations, such as binding to port 921. This is used to limit the damage an intruder could do if he gains control of the lwresd process.

–v Reports the version number, and exits.

In most cases, the lwresd command is run without any options. The defaults are correct for most lwresd configurations. And remember, lwresd is not needed for most systems. Most applications use the traditional stub resolver.

Configuring a Domain Name Server

There are three basic name server configurations:

- A *caching server* is a non–authoritative server. It gets all of its answers to name–server queries from other name servers.
- The *slave server* is considered an authoritative server because it has a complete and accurate domain database that it transfers from the master server. It is also called the *secondary server* because it is the backup for the primary server.
- The *master server* is the primary server for the domain. It loads the domain information directly from a local disk file maintained by the domain administrator. The master server is considered authoritative for the domain, and its answers to queries are always considered to be accurate.

Note Most servers combine elements of more than one configuration. All servers cache answers, and many primary servers are secondary for some other domain. Mix and match these server configurations on your server as needed for your network.

Create only one master server for your domain. It is the ultimate source of information about your domain. Creating more than one master could undermine the reliability of the data. Create at least one slave server. It shares the workload, and provides backup for the master server. Most domain administrators create two official slave servers to increase reliability. Use caching servers throughout the network to reduce the load on the master and secondary servers, and to reduce network traffic by placing name servers close to your users. To enhance performance, many network administrators place a caching server on each subnet or in each department.

Up to five different types of files are required for a named configuration. All configurations require these three basic files:

named configuration file The named.conf configuration file defines basic parameters, and points to the sources of domain database information, which can be local files or remote servers.

hints file The hints, or cache, file provides the names and addresses of the root servers that are used during startup.

local host file All configurations have a local domain database for resolving the loopback address to the hostname localhost.

The other two files that are used to configure named are used only on the master server. These are the two files that define the domain database:

zone file The zone file defines most of the information. It is used to map hostnames to addresses, to identify the mail servers, and to provide a variety of other domain information.

reverse zone file The reverse zone file maps IP addresses to hostnames, which is exactly the opposite of what the zone file does.

Note A *zone* is the piece of the domain namespace over which a master server has authority. The domain database file that contains the information about the zone is called a *zone file*. A zone and a domain are not equivalent. For example, everything in the database file is in a single zone, even if the file contains information about more than one domain.

To configure DNS, you need to understand how to configure all five configuration files. Let's start by looking at the named.conf file. It is used on every name server, and defines the basic configuration.

The *named* Configuration File

When BIND 8 was introduced, everything about the named configuration file changed: its name, the commands it contains, the structure of the commands, and the structure of the file. Administrators familiar with configuring the previous version of BIND were forced to start from scratch. But the introduction of BIND 9 has been less traumatic. The BIND 9 named.conf file has the same structure, a similar syntax, and only two additional commands.

The structure of the named.conf configuration commands is similar to the structure of the C programming language. A statement ends with a semicolon (;), literals are enclosed in quotes ("), and related items are grouped together inside curly braces ({}). BIND provides three ways to insert a comment. A comment can be enclosed between /* and */, like a C language comment. It can begin with two slashes (//), like a C++ comment; or it can begin with a hash mark (#), like a shell comment. The examples in this book use C++ style comments, but you can use any of the three valid styles that you like. The complete syntax of each command is covered in Appendix B, "BIND Reference."

There are eleven valid configuration statements for BIND 9.1, which is the version of BIND delivered with Red Hat 7.2. They are listed alphabetically in Table 4.1 with a short description of each command.

Table 4.1: *named.conf* Configuration Statements

Command	Usage
acl	Defines an access control list
controls	Defines the control channel for the named control program
include	Includes another file into the configuration file
key	Defines security keys for authentication
logging	Defines what will be logged and where it will be stored
lwres	Causes the server to act as a lightweight resolver server (BIND 9 only)

options	Defines global configuration options and defaults
server	Defines a remote server's characteristics
trusted–keys	Defines the DNSSEC encryption keys for the server
view	Shows different views of the zone data to different clients (BIND 9 only)
zone	Defines a zone

The next few sections use examples to illustrate the function and format of the most commonly used commands. This chapter is a tutorial that focuses on the common configurations used on operational networks. Appendix B gives the syntax of all commands, even those that are rarely used. Additionally, "Linux DNS Server Administration," part of the Craig Hunt Linux Library from Sybex, is a book–length treatment of DNS for readers who want even more examples.

The *options* Statement

Most named.conf files open with an options statement. Only one options statement can be used. The options statement defines global parameters that affect the way BIND operates. It also sets the defaults used by other statements in the configuration file. The most commonly used options statement defines the working directory for the server:

```
options {
        directory "/var/named"
};
```

The statement starts with the options command. The curly braces enclose a list of options, and the keyword directory defines the directory that named will read files from and write files to. The directory name is also used to complete any filename specified in the configuration file. The literal enclosed in quotes is the pathname of the directory. Notice that both the directory clause and the options statement end with a semicolon.

As you'll see in the section on the named control tool later in this chapter, named writes several different files that are used to check the status of the name server. The options statement can be used to change the default pathnames of the individual files. However, it is generally best to keep the standard name for each status file and to store these files in the directory identified by the directory command. Doing so makes things simpler for others who attempt to diagnose a name server problem on your system.

Several options can be set that affect all zones or all servers. In most cases, it is better to set those values specifically for the zone or server that is being affected. The designers of BIND have set the defaults correctly for the vast majority of zones and servers. Zones and servers that need other values are exceptions, and they should be treated as such by defining the exceptional characteristics directly on the zone or server statement—not in the options statement. You'll see examples of defining options in the zone statement in the next section.

One option used on occasion is forwarders. The forwarders option causes the local server to forward to a specific list of servers all queries that it cannot resolve from its own cache. This builds a rich cache on the selected servers. The selected servers should be on your local network because the point of building the rich cache is to reduce the number of queries sent over the wide area network. This is primarily useful if your WAN charges for usage, as some ISDN networks do.

A sample forwarders option is

```
options {
```

```
        directory "/var/name";
        forward first;
        forwarders { 172.16.5.1; 172.16.12.1; };
};
```

The forward first option says that the local server should try the forwarders before it tries to get an answer from any other external server. This is the default, so this option really didn't need to be specified. The other possible value is forward only, which tells the server that it can talk only to the forwarders. Even if the forwarders are not responding, the local server is not allowed to find the answer itself when forwarders only is specified. The forwarders option lists the addresses of the servers to which queries are forwarded.

The *zone* Statement

The zone statements are the most important statements in the configuration file, and they constitute the bulk of the named.conf file. A zone statement performs the following critical configuration functions:

- It defines a zone that is serviced by this name server.
- It defines the type of name server that this server is for the zone. A server can be a master server or a slave server. And because this is defined on a per–zone basis, the same server can be the master for some zones while being a slave for others.
- It defines the source of domain information for a zone. The domain database can be loaded from a local disk file or transferred from the master server.
- It defines special processing options for the zone.

A sample zone statement illustrates all of these functions:

Listing 4.3: A Sample *zone* Statement

```
zone "foobirds.org" in {
        type master;
        file "foobirds.hosts";
        check-names fail;
        notify yes;
        also-notify { 172.16.80.3; };
        allow-update { none; };
};
```

The statement begins with the zone command. The name of the zone is written as a literal enclosed in quotes. The in keyword means that this zone contains IP addresses and Internet domain names. This is the default, so in is not really required. The curly braces enclose a list of options for this zone.

The type master option says that this server is the master server for the foobirds.org domain. Other possible values are

slave Identifies this as a slave server for the domain.

hints Identifies this as the hints file that is used to initialize the name server during startup.

stub Identifies this as a stub server. Stub servers are slave servers that only load

the name server records from the master server's database. This is primarily for non–recursive servers that want to refer a query to another server in the same way that root servers refer questions to other servers. This is rarely used on an operational enterprise network.

The file "foobirds.hosts" option points to the file that contains the domain database information. For a master server, this is the file that is created by the domain administrator.

The last four options are in the example primarily to illustrate how options are used with zones. The check–names fail option specifies what the server should do if it finds invalid hostnames in the zone file. The default is for a master server to abort loading the zone file and display an error message. (Because the default was chosen, this option wasn't actually needed.) Alternative values for this option are warn and ignore. warn displays a warning message, and loads the zone anyway, which is the default for slave servers. ignore just ignores any errors.

The notify and also–notify options determine whether the master server notifies slave servers when the zone information is updated. Slave servers periodically check the master zone file to see whether it has been updated. With the notify yes option, the master server sends a DNS NOTIFY message to the slave servers to cause them to immediately check the zone file. This is done to keep the master and slave databases tightly synchronized. The notify yes option is the default, so it didn't need to be specified.

Normally, the DNS NOTIFY message is only sent to official name servers that are listed in the zone file. The also–notify { 172.16.80.3; } option tells the master server to also notify 172.16.80.3. This system is not an official slave server, but it keeps a full copy of the zone database for some other purpose.

The allow–update option specifies which clients are allowed to dynamically update the zone file. In Listing 4.3, allow–update is set to none, meaning that dynamic updates will not be allowed. Because this is the default setting, the allow–update line in Listing 4.3 is not required. Alternatively, the IP addresses of systems allowed to perform dynamic updates could be listed inside the curly braces of the allow–update statement.

The biggest problem with the allow–update option is that it grants a powerful and dangerous privilege on the basis of nothing more than an IP address. As everyone knows who has ever changed the address of a network interface with the ifconfig command, it is very easy to make a Linux system appear to be any address you wish. Trusting an IP address doesn't really provide any security. This potentially dangerous power must be as tightly limited as possible. When using allow–update, only the DHCP server should be allowed to perform updates, and the only acceptable way to allow a Linux DHCP server to dynamically update a Linux DNS server is to run them on a same host. The following allow–update command limits dynamic updates to processes running on the name server:

```
allow-update { localhost; };
```

Prior to BIND 9, allow–update was the only way to secure dynamic updates. BIND 9 adds the update–policy option, which is a new zone statement option that grants dynamic update privileges based on the cryptographic signature of the update packet. update–policy and allow–update are mutually exclusive. You can use one or the other in a zone statement, but not both. Using update–policy to allow dynamic updates from DHCP requires a DHCP server that supports dynamic DNS (DDNS) and provides cryptographic signatures. DHCP is covered in Chapter 8, "Desktop Configuration Servers." To read more about DDNS, DNSSEC, and the use of cryptographic signatures in DNS, see *Linux DNS Server Administration* by Craig Hunt, Sybex, 2001.

In most named.conf files, the zone statements are simpler than the example shown above. Let's look at some more realistic examples of name server configurations.

A Caching–Only Configuration

The *caching–only configuration* is the foundation of all server configurations because all servers cache answers. The most common caching–only configuration is shown in Listing 4.4. This name.conf file is based on the sample found in the BIND 9 documentation with slight modifications for our imaginary network.

Listing 4.4: A Common Caching–Only Configuration

```
$ cat /etc/named.conf
//
// accept queries from the local subnet
//
acl "subnet12" { 172.16.12.0/24; };
options {
        directory "/var/named";
        allow-query { "subnet12"; };
};

//
// a caching only nameserver config
//
zone "." {
        type hint;
        file "named.ca";
};

zone "0.0.127.in-addr.arpa" {
        type master;
        file "named.local";
};
```

The named.conf file in Listing 4.4 opens with an access control list. Use the acl command to assign an arbitrary name to a list of items that will be subsequently referenced by that name. In Listing 4.4, the acl command assigns the name subnet12 to a list of addresses. In this case, the list contains only one network address, but it could have contained more. The name subnet12 is then referenced in the allow–query option in the options statement.

The allow–query option limits the clients from which DNS queries will be accepted. By default, a BIND server accepts queries from any source. However, a caching–only server is not advertised to the outside world, and in general is intended to service only a limited number of local systems. The allow–query option in Listing 4.4 ensures that this server will only provide service to the clients on network 172.16.12.0.

The directory option in the options statement defines the default directory for named. In the sample file, it is /var/named. All subsequent file references in the named.conf file are relative to this directory.

The two zone statements in this caching–only configuration are found in all server configurations. The first zone statement defines the hints file that is used to help the name server locate the root servers during startup. The second zone statement makes the server the master for its own loopback address, and points to the local host file.

The Red Hat Caching–Only Configuration

The caching–only configuration is the most common DNS server configuration; so common, in fact, that many systems are delivered with a ready–made, caching–only server configuration. Red Hat provides a caching–only configuration in RPM format. Figure 4.1 shows a Gnome RPM query for the Red Hat package containing the caching–only server configuration.

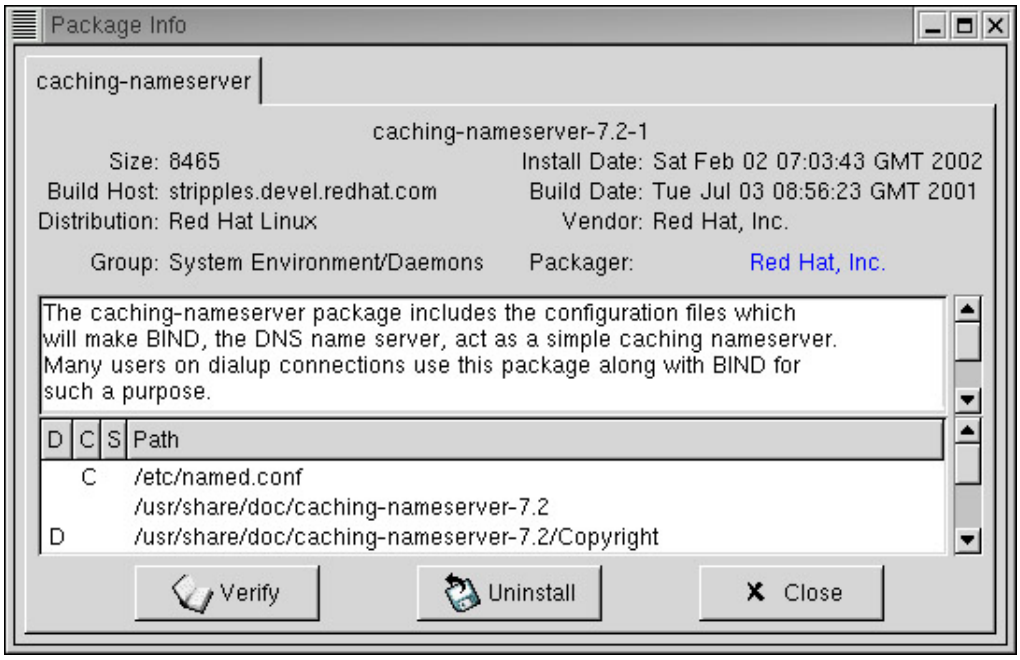


Figure 4.1: A caching–only DNS server RPM

Installing the caching–nameserver–7.2–1 RPM creates the named.conf file shown in Listing 4.5.

Listing 4.5: The Red Hat *named.conf* File

```
// generated by named-bootconf.pl

options {
        directory "/var/named";
        /*
         * If there is a firewall between you and nameservers you want
         * to talk to, you might need to uncomment the query-source
         * directive below.  Previous versions of BIND always asked
         * questions using port 53, but BIND 8.1 uses an unprivileged
         * port by default.
         */
        // query-source address * port 53;
};

//
// a caching only nameserver config
//
zone "." IN {
        type hint;
        file "named.ca";
};

zone "localhost" IN {
type master;
        file "localhost.zone";
        allow-update { none; };
```

```
};

zone "0.0.127.in-addr.arpa" IN {
        type master;
        file "named.local";
        allow-update { none; };
};
key "key" {
        algorithm hmac-md5;
        secret "eabDFqxVnhWyhUwoSVjthOue0bYtvQUCiSuBqHxDRWilSaWMoMORNLmyEbJr";
};
```

Most of the configuration commands in this named.conf file are commands we have already seen. The directory option, the zone statement for the "." zone, and the zone statement for the "0.0.127.in–addr.arpa" zone have all been covered. But there are a few new items.

The first is the query–source option, which is commented out of the options statement at the beginning of the file. The comment implies that this might be needed if you have a firewall. Of course, anything is possible with a firewall, but it is unlikely that this option will be needed. By default, BIND 8 and BIND 9 servers send queries to remote servers using the well–known port 53 as the destination port and a randomly generated, non–privileged port as the source. This is exactly how most TCP/IP services operate—they use well–known ports as destination ports and randomly generated, non–privileged ports as the source. Most firewalls do not block out bound traffic that originates on a non–privileged port; if they did, they would block essentially all outbound traffic. If, for some reason, your firewall blocks outbound DNS traffic when it has a non–privileged source port, use the query–source option shown in Listing 4.5 to send queries using the address of the server and a source port number of 53.

Another new item in this configuration is the key statement. Use the key statement to assign a key identifier to the algorithm and secret key pairing that will be used for transaction security. Transaction security guarantees the authenticity and integrity of the DNS queries and responses that move over the network. A query and the response to that query is a *transaction*. The authenticity and integrity of DNS transactions are guaranteed by digital signatures, which in the case of transactions are called *transaction signatures* (TSIGs). A TSIG is not a DNS database record; it is a technique for digitally signing DNS messages. The key statement defines the algorithm and secret key used to digitally sign transactions. The syntax of the key statement is

key *key_id* {

  algorithm hmac–md5;

  secret *secret_string;*

};

The *key_id* is any descriptive name you want to assign to the algorithm and secret key pairing— it is this name that is used as the key identifier. In Listing 4.5, the algorithm/key pairing is simply named "key". Currently, BIND only supports the hmac–md5 algorithm, so the algorithm is always that value (although this could change in the future). The *secret_string* is a base–64 encoded key used by the algorithm. Use the dnskeygen utility to generate the *secret_string* value, or manually create your own value if you have another utility that can create encoded keys.

After a key identifier is defined, it can be used in an access control list or in the keys option. Notice

that the key defined in the example is not referenced anywhere in Listing 4.5. The Red Hat configuration creates this key for use with the remote named control tool. Later in this chapter, we will see this same key in the Red Hat rndc.conf file that is used to configure the remote named control tool.

The most unique item in this configuration is the zone statement for a zone named "localhost". The Red Hat server is the master for the "localhost" zone, so it loads the zone directly from a local file. The file is named /var/named/localhost.zone. Listing 4.6 shows this file.

Listing 4.6: The Red Hat *localhost.zone* File

```
$ cat /var/named/localhost.zone
$TTL    86400
$ORIGIN localhost.
@                1D IN SOA       @ root (
                                 42              ; serial (d. adams)
                                 3H              ; refresh
                                 15M             ; retry
                                 1W              ; expiry
                                 1D )            ; minimum

                 1D IN NS        @
                 1D IN A         127.0.0.1
```

The purpose of this file is to map the domain name localhost. to the address 127.0.0.1. In most cases, it is unused. Generally, users query for the hostname localhost, not the hostname localhost.. When the name is entered without the trailing dot, it is expanded with the domain names from the search list and sent to the server for the local domain. The name localhost is either resolved on the local system by the small /etc/hosts file found on most systems, or sent to the name server of the local domain for resolution. Only the name localhost. with the trailing dot will not be expanded and thus be resolved by this file. However, this file provides a nice balance to the reverse zone file that Red Hat provides for the loopback address because that file makes multiple references to the filename localhost.. Most systems do not have a localhost.zone file. Red Hat administrators use this file because Red Hat has already created it for them.

Most Unix and Linux systems do not create the "localhost" zone. But all server configurations create the hints file and the reverse zone file for the loopback address. It is the loopback address reverse zone file that is most commonly called the local host file. The hints file and the local host file, along with the named.conf file, are required for every server configuration.

The Hints File

The hints file contains information that named uses to initialize the cache. As indicated by the root domain (".") name on the zone statement, the hints the file contains are the names and addresses of the root name servers. The file helps the local server locate a root server during startup. After a root server is located, an authoritative list of root servers is downloaded from that server. The hints are not referred to again until the local server restarts.

The named.conf file points to the location of the hints file. The hints file can be given any filename. Commonly used names are named.ca, named.root, and root.cache. In the Red Hat example, the hints file is called named.ca, and is located in the /var/named directory. The hints file provided by the Red Hat installation contains the following server names and addresses:

Listing 4.7: The *named* Hints File

```
.                    3600000  IN  NS  A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.  3600000      A   198.41.0.4
.                    3600000      NS  B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET.  3600000      A   128.9.0.107
.                    3600000      NS  C.ROOT-SERVERS.NET.
C.ROOT-SERVERS.NET.  3600000      A   192.33.4.12
.                    3600000      NS  D.ROOT-SERVERS.NET.
D.ROOT-SERVERS.NET.  3600000      A   128.8.10.90
.                    3600000      NS  E.ROOT-SERVERS.NET.
E.ROOT-SERVERS.NET.  3600000      A   192.203.230.10
.                    3600000      NS  F.ROOT-SERVERS.NET.
F.ROOT-SERVERS.NET.  3600000      A   192.5.5.241
.                    3600000      NS  G.ROOT-SERVERS.NET.
G.ROOT-SERVERS.NET.  3600000      A   192.112.36.4
.                    3600000      NS  H.ROOT-SERVERS.NET.
H.ROOT-SERVERS.NET.  3600000      A   128.63.2.53
.                    3600000      NS  I.ROOT-SERVERS.NET.
I.ROOT-SERVERS.NET.  3600000      A   192.36.148.17
.                    3600000      NS  J.ROOT-SERVERS.NET.
J.ROOT-SERVERS.NET.  3600000      A   198.41.0.10
.                    3600000      NS  K.ROOT-SERVERS.NET.
K.ROOT-SERVERS.NET.  3600000      A   193.0.14.129
.                    3600000      NS  L.ROOT-SERVERS.NET.
L.ROOT-SERVERS.NET.  3600000      A   198.32.64.12
.                    3600000      NS  M.ROOT-SERVERS.NET.
M.ROOT-SERVERS.NET.  3600000      A   202.12.27.33
```

The hints file contains only name server (NS) and address (A) records. Each NS record identifies a name server for the root domain (.). The associated A record gives the IP address for each server. The structure of these database entries will become clear later in the chapter. For now, it is important to realize that you do not directly create or edit this file.

The sample file in Listing 4.7 is provided by the Linux installation. But even if your system doesn't provide a hints file, it is easy to get one. The official list of root servers is available on the Internet. Download the file /domain/named.root from ftp://ftp.rs.internic.net/ via anonymous FTP. The file that is stored there is in the correct format for a Linux system, is ready to run, and can be downloaded directly to your hints file.

The Local Host File

Every name server is the master of its own loopback domain, which of course makes sense. The whole point of creating the loopback interface (lo) is to reduce network traffic. Sending domain queries about the loopback address across the network would defeat that purpose.

The loopback domain is a reverse domain. It is used to map the loopback address 127.0.0.1 to the hostname localhost. On our sample Red Hat system, the zone file for this domain is called named.local, which is the most common name for the local host file. The Red Hat installation provides the file shown in Listing 4.8.

Listing 4.8: The *named.local* File

```
$ cat /var/named/named.local
@       IN      SOA     localhost. root.localhost.  (
                        1997022700 ; Serial
```

```
                        28800      ; Refresh
                        14400      ; Retry
                        3600000    ; Expire
                        86400 )    ; Minimum
        IN      NS      localhost.

1       IN      PTR     localhost.
```

Every Linux system that runs named has an essentially identical local host file. This one was created automatically by the Red Hat installation; if your system doesn't create one, you can copy this one. There is really no need to edit or change this file to run it on your system. At this point, the contents of the file don't need to be discussed because they are always the same on every system. You will, however, see examples of all of these database records later in the chapter.

Most name servers are caching–only servers. For those servers, you:

- Configure the resolver. When running named on the local system, you can use the default resolver configuration.
- Create the named.conf file. You can copy the one shown in Listing 4.4.
- Download the named.root from ftp://ftp.rs.internic.net/, and use it as the hints file.
- Create the named.local file. You can copy the one shown in Listing 4.8.
- Start named, and add named to the system startup to ensure that named starts automatically whenever the system reboots. See "Running named" at the end of this chapter for more information on starting named.

This simple configuration works on most name servers, but not on all of them. The slave servers and the master server require more effort.

The Slave Server Configuration

Configuring a slave server is almost as simple as configuring a caching–only server. It uses the same three configuration files with only minor modifications to the named.conf file. Because of this, you can start with a caching–only configuration to test your system before you configure it as a slave server. Our sample slave server will be built by modifying the common caching–only configuration shown in Listing 4.4.

Assume that wren (172.16.5.1) is the master server for the foobirds.org domain and the 16.172.in–addr.arpa reverse domain, and that we want to configure falcon as a slave server for those domains. To accomplish this, we add two new zone statements to the basic named.conf file on falcon to create the configuration shown in Listing 4.9.

Listing 4.9: A DNS Slave Server Configuration

```
$ cat /etc/named.conf
options {
        directory "/var/named";
};

// a slave server configuration
//
zone "." {
        type hint;
        file "named.ca";
};
```

```
zone "0.0.127.in-addr.arpa" {
        type master;
        file "named.local";
};

zone "foobirds.org" {
        type slave;
        file "foobirds.hosts";
        masters { 172.16.5.1; };
        allow-updates { none; };
};

zone "16.172.in-addr.arpa" {
        type slave;
        file "172.16.reverse";
        masters { 172.16.5.1; };
        allow-updates { none; };
};
```

The access control list and the allow–query option from Listing 4.4 have been removed from this configuration. Authoritative servers (the master and all official slaves) should accept queries from any source because they are advertised to the world through the domain's NS records. When you advertise a service, you should provide it.

The configuration file contains all of the zone statements that have already been discussed because all servers use a hints file and a loopback domain database file. The two new zone statements declare zones for the domains foobirds.org and 0.16.172.in–addr.arpa. The type clause in each of the new zone statements says that this is a slave server for the specified domains.

The file clause for a slave zone has a different purpose than those that you have seen before. In the previous examples, the file identified by the file clause was the source of the zone information. In this case, the file is the local storage for the zone information. The ultimate source for the information is the master server.

The masters clause identifies the master server. There can be more than one IP address provided in this clause, particularly if the master server is multihomed and thus has more than one IP address. In most configurations, only one address is used, which is the address of the master server for the specified zone.

The slave server downloads the entire zone file from the master server. This process is called a *zone file transfer*. When the file is downloaded, it is stored in the file identified by the file clause. Don't create or edit this file; it is created automatically by named. After the zone is downloaded, it loads directly from the local disk. The slave will not transfer the zone again until the master server updates the zone. How the slave knows when the zone has been updated is covered later in this chapter.

Configuring caching servers and slave servers doesn't seem very difficult, so what's the big deal about DNS configuration? The big deal is the master server, and that's what we tackle next.

The Master Server Configuration

The named.conf file for a master server looks very much like the configuration file for a secondary server. In Listing 4.9, falcon was the slave server for foobirds.org and 16.172.in–addr.arpa, and

wren was the master server for those domains. The named.conf file for wren is shown in Listing 4.10.

Listing 4.10: A DNS Master Server Configuration

```
$ cat /etc/named.conf
options {
        directory "/var/named";
};

// a master nameserver config
//
zone "." {
        type hint;
        file "named.ca";
};

zone "0.0.127.in-addr.arpa" {
        type master;
        file "named.local";
};

zone "foobirds.org" {
        type master;
        file "foobirds.hosts";
        notify yes;
        allow-updates { none; };
};

zone "16.172.in-addr.arpa" {
        type master;
        file "172.16.reverse";
        notify yes;
        allow-updates { none; };
};
```

The zone statements for the foobirds.org and 16.172.in–addr.arpa domains are almost the same as the zone statement for the 0.0.127.in–addr.arpa domain, and they function the same way: The statements declare the zones, say that this is the master server for those zones, and identify the files that contain the database records for those zones. The new zone statements also have two options. notify was added so that the server will send DNS NOTIFY messages to the slave servers whenever the zone file is updated. allow–update is set to none to reject dynamic updates.

So far, the configuration of the master server is the same as any other server—you create a configuration file, a hints file, and a local host reverse zone file. The difference comes from the fact that you must also create the real domain database files. The foobirds.hosts file and the 172.16.reverse file in our example can't be downloaded from a repository. You must create them, and in order to do so, you must understand the syntax and purpose of the database records.

DNS Database Records

The database records used in a zone file are called *standard resource records* or sometimes just *RRs*. All resource records have the same basic format:

```
[name] [ttl] IN type data
```

The name field identifies the domain object affected by this record. It could be an individual host or an entire domain. Unless the name is a fully qualified domain name, it is relative to the current domain.

A few special values can be used in the name field. These are

> A blank name refers to the last named object. The last value of the name field stays in force until a new value is specified.
>
> @ An at−sign refers to the current *origin*. The origin is the default domain name used inside the zone file. You can set the origin in the database file with the $ORIGIN directive. If a $ORIGIN directive is not used, the origin is the domain name from the zone command in the named.conf file.
>
> * An asterisk is a wildcard character that can be used to match any character string.

The time−to−live (ttl) field defines the length of time that this resource record should be cached. This permits you to decide how long remote servers should store information from your domain. You can use a short TTL for volatile information and a long TTL for stable information. If no TTL value is specified, the default TTL value defined by the $TTL directive is used. (Zone file directives are discussed later in the chapter.)

The class field is always IN, which is shown in the syntax above. There really are three possible values: HS for Hesiod servers, CH for Chaosnet servers, and IN for Internet servers. All of the information you deal with is for TCP/IP networks and Internet servers, so you will not use the other values.

The type field defines the type of resource record. There are 40 different types of records; almost all of which are experimental, obsolete, or unused. The types used in this chapter, which are the most commonly used record types, are listed in Table 4.2.

Table 4.2: DNS Database Record Types

Record Name	Record Type	Function
Start of Authority	SOA	Marks the beginning of a zone's data, and defines parameters that affect the entire zone
Name Server	NS	Identifies a domain's name server
Address	A	Maps a hostname to an address
Pointer	PTR	Maps an address to a hostname
Mail Exchanger	MX	Identifies the mail server for a domain
Canonical Name	CNAME	Defines an alias for a hostname

The last field in the resource record is the data field, which holds the data that is specific to the type of resource record. For example, in an A record, this contains an address. The format and function of the data field is different for every record type.

In addition to resource records, BIND provides four zone file directives that are used to simplify the construction of the zone file or to define a value used by the resource records in the file.

Zone File Directives

The four directives are evenly divided into two that simplify the construction of a zone file, $INCLUDE and $GENERATE; and two that define values used by the resource records, $ORIGIN and $TTL.

The $TTL Directive Defines the default TTL for resource records that do not specify an explicit time to live. The TTL value can be specified as a number of seconds, or as a combination of numbers and letters. Defining one week as the default TTL using seconds is

```
$TTL 604800
```

Using the alphanumeric format, one week can be defined as

```
$TTL 1w
```

The letter values that can be used with the alphanumeric format are

- w for week
- d for day
- h for hour
- m for minute
- s for second

The $ORIGIN Directive Sets the current origin, which is the domain name used to complete any relative domain names. By default, $ORIGIN starts out assigned the domain name defined on the zone statement. Use the $ORIGIN directive to change the setting. For example, the following directive changes the origin to ducks.foobirds.org:

```
$ORIGIN ducks.foobirds.org.
```

Like all names in a zone file, the domain name on the $ORIGIN directive is relative to the current origin, unless it is fully qualified. Thus, if the zone statement defines the domain foobirds.org and the zone file contains the following $ORIGIN directive:

```
$ORIGIN ducks
```

The effect is the same as the previous $ORIGIN directive. The name ducks is relative to the current origin foobirds.org. Therefore, the new origin is ducks.foobirds.org. Relative names in any resource records in the zone file that follow this $ORIGIN directive are relative to this new origin.

The $INCLUDE Directive Reads in an external file, and includes it as part of the zone file. The external file is included in the zone file at the point where the $INCLUDE directive occurs. The $INCLUDE directive makes it possible to divide a large domain into several different files. This might be done so that several different administrators can work on various parts of a zone without having all of them try to work on one file at the same time. The directive begins with the $INCLUDE keyword, which is followed by the name of the file to be included. All filenames in the zone file are relative to the directory pointed to by the directory option in the named.conf file, unless fully qualified to the root.

The $GENERATE Directive The $GENERATE directive create a series of resource records. The resource records generated by the $GENERATE directive are almost identical, varying only by a numeric iterator. An example shows the structure of the $GENERATE directive:

```
$ORIGIN 20.16.172.in-addr.arpa.
$GENERATE 1-4 $ CNAME $.1to4
```

The $GENERATE keyword is followed by the range of records to be created. In the example, the range is 1 through 4. The range is followed by the template of the resource records to be generated. In this case, the template is $ CNAME $.1to4. A $ sign in the template is replaced by the current iterator value. In the example, the value iterates from 1 to 4. This $GENERATE directive produces the following resource records:

```
1 CNAME 1.1to4
2 CNAME 2.1to4
3 CNAME 3.1to4
4 CNAME 4.1to4
```

Given that 20.16.172.in–addr.arpa. is the value defined for the current origin, these resource records are the same as

```
1.20.16.172.in-addr.arpa. CNAME 1.1to4.20.16.172.in-addr.arpa.
2.20.16.172.in-addr.arpa. CNAME 2.1to4.20.16.172.in-addr.arpa.
3.20.16.172.in-addr.arpa. CNAME 3.1to4.20.16.172.in-addr.arpa.
4.20.16.172.in-addr.arpa. CNAME 4.1to4.20.16.172.in-addr.arpa.
```

These odd–looking records have a very specific purpose for delegating reverse subdomains. Delegating reverse domains is covered later in this chapter. The purpose of these odd resource records is described there.

Now that you know what records and directives are available and what they look like, you're ready to put them together to create a database.

The Domain Database File

The domain database file contains most of the domain information. Its primary function is to convert hostnames to IP addresses so A records predominate, but this file contains all of the database records except PTR records. Creating the domain database file is both the most challenging and the most rewarding part of building a name server.

In the foobirds.org domain, wren is the master server. Based on the named.conf file shown in Listing 4.10, the domain database file is named foobirds.hosts. Its contents are shown in Listing 4.11.

Listing 4.11: A Sample DNS Zone File

```
;
;      The foobirds.org domain database
;
$TTL 1w
@    IN  SOA   wren.foobirds.org. sara.wren.foobirds.org. (
               2002030601 ; Serial
               21600      ; Refresh
```

```
                1800       ; Retry
                604800     ; Expire
                900 )      ; Negative cache TTL
;  Define the nameservers
                 IN      NS      wren.foobirds.org.
                 IN      NS      falcon.foobirds.org.
                 IN      NS      bear.mammals.org.
;  Define the mail servers
                 IN      MX      10 wren.foobirds.org.
                 IN      MX      20 parrot.foobirds.org.
;
;       Define localhost
;
localhost        IN      A       127.0.0.1
;
;       Define the hosts in this zone
;
wren             IN      A       172.16.5.1
parrot           IN      A       172.16.5.3
crow             IN      A       172.16.5.5
hawk             IN      A       172.16.5.4
falcon           IN      A       172.16.5.20
puffin           IN      A       172.16.5.17
                 IN      MX      5 wren.foobirds.org.
robin            IN      A       172.16.5.2
                 IN      MX      5 wren.foobirds.org.
redbreast        IN      CNAME   robin.foobirds.org.
www              IN      CNAME   wren.foobirds.org.
news             IN      CNAME   parrot.foobirds.org.
;
;       Delegating subdomains
;
swans          IN      NS      trumpeter.swans.foobirds.org.
               IN      NS      parrot.foobirds.org.
terns          IN      NS      arctic.terns.foobirds.org.
               IN      NS      trumpeter.swans.foobirds.org.
;
;    Glue records for subdomain servers
;
trumpeter.swans     IN      A       172.16.12.1
arctic.terns         IN      A       172.16.6.1
```

The SOA record All zone files begin with an SOA record. The @ in the name field of the SOA record refers to the current origin, which in this case is foobirds.org because that is the value defined in the zone statement of the configuration file. Because it ties the domain name back to the named configuration file, the name field of the SOA record is usually an at–sign.

The data field of the SOA record contains seven different components. It is so long, the data field of the SOA record normally spans several lines. The parentheses are continuation characters. After an opening parenthesis, all data on subsequent lines are considered part of the current record until a closing parenthesis. The components of the data field in the sample SOA record contain the following values:

wren.foobirds.org This is the hostname of the master server for this zone.

sara.wren.foobirds.org This is the e–mail address of the person responsible for this domain. Notice that the at–sign (@) normally used between the username (sara) and the hostname (wren.foobirds.org) in an e–mail address is replaced here with a

dot (.).

2002030601 This is the serial number, a numeric value that tells the slave server that the zone file has been updated. To make this determination, the slave server periodically queries the master server for the SOA record. If the serial number in the master server's SOA record is greater than the serial number of the slave server's copy of the zone, the slave transfers the entire zone from the master. Otherwise, the slave assumes it has a current copy of the zone, and skips the zone transfer. The serial number should be increased every time the domain is updated in order to keep the slave servers synchronized with the master.

21600 This is the length of time in the refresh cycle. Every refresh cycle, the slave server checks the serial number of the SOA record from the master server to determine whether the zone needs to be transferred. The length of the refresh cycle can be defined using either a numeric or an alphanumeric format. (See the discussion of the $TTL directive for the details of these formats.) Listing 4.11 uses the numeric format to set the refresh cycle to 21,600 seconds, which tells the slave server to check four times per day. This indicates a stable database that does not change very frequently, which is often the case. Computers are added to the network periodically, but not usually on an hourly basis. When a new computer arrives, the hostname and address are assigned before the system is added to the network because the name and address are required to install and configure the system. Thus, the domain information is disseminated to the slave servers before users begin to query for the address of the new system. A low refresh cycle keeps the servers tightly synchronized, but a very low value is not usually required because the DNS NOTIFY message sent from the master server causes the slave to immediately check the serial number of the SOA record when an update occurs. The refresh cycle is a redundant backup for DNS NOTIFY.

1800 This is the retry cycle. The retry cycle defines the length of time that the slave server should wait before asking again when the master server fails to respond to a request for the SOA record. The length of time can be specified using either a numeric or an alphanumeric format. In this example, the numeric format is used to specify a retry time of 1800 seconds (30 minutes). Don't set the value too low—a half–hour or 15 minutes are good retry values. If the server doesn't respond, it may be down. Quickly retrying a down server gains nothing, and wastes network resources.

604800 This is the expiration time, which is the length of time that the slave server should continue to respond to queries, even if it cannot update the zone file. The idea is that at some point in time, out–of–date data are worse than no data. This should be a substantial amount of time. After all, the main purpose of a slave server is to provide backup for the master server. If the master server is down and the slave stops answering queries, the entire network is down instead of having just one server down. A disaster, such as a fire in the central computer facility, can take the master server down for a very long time. The time can be specified in either numeric or alphanumeric format. In Listing 4.10, 604,800 seconds (one week) is used; an equally common value is one month.

900 This is the default time–to–live that servers should use when caching negative information about this zone. All servers cache answers, and use those answers to respond to subsequent queries. Most of the answers cached by a server are

standard resource records, which is positive information from the DNS database. A name server can also learn from the authoritative server for a zone that a specific piece of information does not exist, which is negative information. For example, the response to a query for bittern.foobirds.org would be that the domain name does not exist. This is valuable information that also should be cached. But with no associated resource record, and thus no explicit TTL, how long should it be cached? The negative cache TTL from the zone's SOA record tells remote servers how long to cache negative information. The SOA record in Listing 4.11 sets the negative cache value to 15 minutes (900 seconds). The negative cache TTL can be defined in either numeric or alphanumeric format. Use a negative cache of no more than 15 minutes—five minutes isn't bad either.

All of the components of the data field of the SOA record set values that affect the entire domain. Several of these items affect remote servers. You decide how often slave servers check for updates, and how long caching servers keep your data in their caches. The domain administrator is responsible for the design of the entire domain.

Defining the Name Servers In Listing 4.11, the NS records that follow the SOA record define the official name servers for the domain. Unless the also−notify option is used in the zone statement of the named.conf file, these are the only servers that receive a DNS NOTIFY message when the zone is updated.

Although they can appear anywhere in the file, the NS records often follow directly after the SOA record. When they do, the name field of each NS record can be blank. Because the name field is blank, the value of the last object named is used. In Listing 4.11, the last value to appear in the name field was the @ that referred to the foobirds.org domain defined in the named.conf file. Therefore, these NS records all define name servers for the foobirds.org domain.

The first two NS records point to the master server wren and the slave server falcon that we configured earlier. The third server is external to our network. Name servers should have good network connections, and slave name servers should have a path to the Internet that is independent from the path used by the master server. This enables the slave server to fulfill its purpose as a backup server, even when the network that the master server is connected to is down. Large organizations may have independent connections for both servers; small organizations usually do not. If possible, find a server that is external to your network to act as a slave server. Check with your Internet Service Provider (ISP); it may offer this as a service to its customers.

Defining the Mail Servers The first two MX records in Listing 4.11 define the mail servers for this domain. The name field is still blank, meaning that these records pertain to the last named object, which in this case is the entire domain. The first MX record says that wren is the mail server for the foobirds.org domain, with a preference of 10. If mail is addressed to *user*@foobirds.org, the mail is directed to wren for delivery.

The second MX record identifies parrot as a mail server for foobirds.org with a preference of 20. The lower the preference number, the more preferred the server. This means that mail addressed to the foobirds.org domain is first sent to wren. Only if wren is unavailable is the mail sent to parrot. parrot acts as a backup for those times when wren is down or offline.

These two MX records redirect mail addressed to the domain foobirds.org, but they do not redirect mail addressed to an individual host. Therefore, if mail is addressed to jay@hawk.foobirds.org, it is delivered directly to hawk; it is not sent to a mail server. This configuration permits people to use e−mail addresses of the form user@domain when they like, or to use direct delivery to an individual

host when they want that. It is a very flexible configuration.

Some systems, however, may not be capable of handling direct delivery e–mail. An example is a Microsoft Windows system that doesn't run an SMTP mail program. Mail addressed to such a system would not be successfully delivered. To prevent this, assign an MX record to the individual host to redirect its mail to a valid mail server.

There are two examples of this in the sample zone file. Look at the resource records for puffin and robin. The address record of each system is followed by an MX record that directs mail to wren. The MX records have a blank name field, but this time they don't refer to the domain. In both cases, the last value in the name field is the name from the preceding address record. It is this name to which the MX record applies. In one case it is puffin, and in the other it is robin. With these records, mail addressed to daniel@puffin.foobirds.org is delivered to daniel@wren.foobirds.org.

The MX record is only the first step in creating a mail server. The MX is necessary to tell the remote computer where it should send the mail, but for the mail server to successfully deliver the mail to the intended user, it must be properly configured.

> **Note** Chapter 5, "Configuring a Mail Server," looks at how sendmail is configured to properly handle the mail.

Defining the Host Information The bulk of the zone file consists of address records that map hostnames to IP addresses. The first address record in the domain database file in Listing 4.11 maps the name localhost.foobirds.org to the loopback address 127.0.0.1. The reason this entry is included in the database has something to do with the way that the resolver constructs queries. Remember that if a hostname contains no dots, the resolver extends it with the local domain. So when a user enters **telnet localhost**, the resolver sends the name server a query for localhost.foobirds.org. Without this entry in the database, the resolver would make multiple queries before finally finding localhost in the /etc/hosts file. The localhost entry is followed by several address entries for individual hosts in the domain.

The only other unexplained records in the section of Listing 4.11 that defines host information are the CNAME records. The first CNAME record says that redbreast is a hostname alias for robin. Aliases are used to map an obsolete name to a current name or to provide generic names such as www and news. Aliases cannot be used in other resource records. Therefore, take care when placing CNAME records in the domain database. You have seen several examples of the fact that a blank name field refers to the previously named object. If the CNAME record is placed improperly, a record with a blank name field can illegally reference a nickname.

For example, the file contains these records for robin:

```
robin           IN      A       172.16.5.2
                IN      MX      5 wren.foobirds.org.
redbreast       IN      CNAME   robin.foobirds.org.
```

A mistake in placing these records could produce the following:

```
robin           IN      A       172.16.5.2
redbreast       IN      CNAME   robin.foobirds.org.
                IN      MX      5 wren.foobirds.org.
```

This would cause named to display the error "redbreast.foobirds.com has CNAME and other data (illegal)" because the MX record now refers to redbreast. Due to the potential for errors, many

domain administrators put the CNAME records together in one section of the file instead of intermingling them with other resource records.

Delegating a Subdomain The final six resource records in Listing 4.11 delegate the subdomains swans.foobirds.org and terns.foobirds.org. The root servers delegated the foobirds.org domain to us. We now have the authority to delegate any domains within the foobirds.org domain that we wish. In the example, two are delegated.

You have complete freedom to create subdomains and hostnames within your domain. Organizations create subdomains for two basic reasons:

- To simplify the management of a large number of hostnames. This reason is easy to understand; it is exactly why DNS was created in the first place. Delegating pieces of the domain spreads the burden of maintaining the system to more people and computers so that no one person or computer is overwhelmed with work.
- To recognize the structure within the organization. This reason springs from a fact of organizational life. Some parts of the organization will want to control their own services, no matter what. In Listing 4.11, two "organizational" subdomains are delegated—one to the group dedicated to research on swans and one to the group that works with terns.

Subdomains usually have either geographic or organizational names. denver.foobirds.org is an example of a geographic subdomain name while sales.foobirds.org is an example of an organizational name. One problem with naming a subdomain is that office locations change, and organizations reorganize. If the subdomain names you choose are too specific, you can bet that you will have to change them. Assume that your west coast office is in Santa Clara. You're better off naming the subdomain west or westcoast than you are calling it santaclara. If they move to a new building in San Jose, you don't want to have to change the subdomain name.

A domain does not officially exist until it is delegated by its parent domain. The administrator of trumpeter.swans.foobirds.org can configure the system as the master server for the swans domain, and enter all of the necessary domain data. It doesn't matter because no one will query the system for information about the domain. In fact, no computer in the outside world will even know that the swans domain exists. When you think of how the domain system works, you'll see why this is true.

The DNS system is a rooted hierarchical system. If a remote server has no information at all about the swans.foobirds.org domain, it asks a root server. The root server tells the remote server that wren and its slave servers know about foobirds.org. The remote server then asks wren. wren finds the answer to the query, either from its cache or by asking parrot or trumpeter, and replies with the answer along with the NS records for swans.foobirds.org and the IP addresses of trumpeter and parrot. Armed with the NS records and the IP addresses, the remote server can send other queries about the swans.foobirds.org domain directly to trumpeter.

The information path is from the root to wren and then to trumpeter. There is no way for the remote server to go directly to trumpeter or parrot for information until wren tells it where they are located. If the delegation did not exist in the foobirds.org domain, the path to the swans.foobirds.org domain would not exist.

Note Notice that the root server sends the remote server to wren, whereas wren looks up the answer for the remote server instead of just sending the remote server to trumpeter. The root servers are *non–recursive* servers: If they don't have an answer, they'll tell you who does, but they won't look it up for you. Most other servers are *recursive* servers: If they don't have the answer, they'll look it up for you.

The first four lines of the sample delegations are NS records.

```
swans          IN      NS      trumpeter.swans.foobirds.org.
               IN      NS      parrot.foobirds.org.
terns          IN      NS      arctic.terns.foobirds.org.
               IN      NS      trumpeter.swans.foobirds.org.
```

The first two records say that trumpeter and parrot are authoritative servers for the swans.foobirds.org domain. The last two records say that arctic and trumpeter are authoritative servers for the terns.foobirds.org domain.

Two other records are part of the subdomain delegation. They are address records:

```
trumpeter.swans     IN      A       172.16.18.15
arctic.terns        IN      A       172.16.6.1
```

Both of these addresses are for name servers located in domains that are subordinate to the current domain. These address records are called *glue records* because they help to link all of the domains together. In order to connect to a name server, you must have its address. If the address for *arctic* were only available from *arctic*, there'd be a problem. For this reason, the address of a name server located in a subordinate domain is placed in the parent domain when the subordinate domain is delegated.

The Reverse Domain File

The reverse domain file maps IP addresses to hostnames. This is the reverse of what the domain database does when it maps hostnames to addresses.

But there is another reason this is called the reverse domain: All of the IP addresses are written in reverse. For example, in the reverse domain, the address 172.16.5.2 is written as 2.5.16.172.in–addr.arpa. The address is reversed to make it compatible with the structure of a domain name. An IP address is written from the most general to the most specific. It starts with a network address, moves through a subnet address, and ends with a host address. The hostname is just the opposite. It starts with the host, moves through subdomain and domain, and ends with a top–level domain. To format an address like a hostname, the host part of the address is written first, and the network is written last. The network address becomes the domain name, and the host address becomes a hostname within the domain.

In our example, the network address 172.16.0.0 becomes the domain 16.172.in–addr.arpa. The zone file for this domain is shown in Listing 4.12.

Listing 4.12: A DNS Reverse Zone File

```
;        Address to hostname mappings.
;
$TTL 1w
@    IN  SOA   wren.foobirds.org. sara.wren.foobirds.org. (
               1999022702   ;   Serial
               21600        ;   Refresh
               1800         ;   Retry
               604800       ;   Expire
               900 )        ;   Negative cache TTL
              IN      NS      wren.foobirds.org.
              IN      NS      falcon.foobirds.org.
              IN      NS      bear.mammals.org.
1.5           IN      PTR     wren.foobirds.org.
```

```
2.5             IN      PTR     robin.foobirds.org.
3.5             IN      PTR     parrot.foobirds.org.
4.5             IN      PTR     hawk.foobirds.org.
5.5             IN      PTR     crow.foobirds.org.
17.5            IN      PTR     puffin.foobirds.org.
20.5            IN      PTR     falcon.foobirds.org.
1.12            IN      PTR     trumpeter.swans.foobirds.org.
1.6             IN      PTR     arctic.terns.foobirds.org.
6               IN      NS      arctic.terns.foobirds.org.
                IN      NS      falcon.foobirds.org.
```

Like other zone files, the reverse zone begins with an SOA record and a few NS records. They serve the same purpose and have the same fields as their counterparts in the domain database, which were explained previously.

PTR records make up the bulk of the reverse domain because they are used to translate addresses to hostnames. Look at the first PTR record. The name field contains 1.5. This is not a fully qualified name, so it is interpreted as relative to the current domain, giving us 1.5.16.172.in–addr.arpa as the value of the name field. The data field of a PTR record contains a hostname. The hostname in the data field is fully qualified to prevent it from being interpreted as relative to the current domain. In the first PTR record, the data field is wren.foobirds.org, so a PTR query for 1.5.16.172.in–addr.arpa (172.16.5.1) returns the value wren.foobirds.org.

Delegating a Reverse Subdomain The last two resource records in Listing 4.12 are NS records that delegate the subdomain 6.16.172.in–addr.arpa to arctic and falcon. Subdomains can be created and delegated in the reverse domain just as they are in the hostname space. However, limitations caused by the way that addresses are treated as hostnames in the reverse domain can make them more difficult to set up and to use than other subdomains.

The reverse domain treats the IP address as a hostname composed of four pieces. The four bytes of the address become the four parts of the hostname. However, addresses are really 32 contiguous bits, not four distinct bytes. In our sample network, we have delegated addresses on byte boundaries, so it's easy to assign reverse subdomains on the same boundary. What if we had only assigned the addresses 172.16.6.1 to 172.16.6.63 to arctic? The delegation shown in Listing 4.12 would send queries to arctic that it couldn't answer.

There is a way around the byte–boundary limitation, but it can be confusing. The technique is to assign every possible address a CNAME that includes a new subdomain that takes into account the real domain structure and then to delegate the new subdomain to the remote server. Assume that we only want arctic to handle the addresses 172.16.6.1 to 172.16.6.63. We could generate 63 CNAME records in the reverse zone as follows:

```
$ORIGIN 6.16.172.in-addr.arpa.
$GENERATE 1-63 $ CNAME $.1-63
1-63        IN      NS        arctic.terns.foobirds.org.
            IN      NS        falcon.foobirds.org.
```

The $GENERATE directive creates 63 CNAME records for the names 1.6.16.172.in–addr.arpa to 63.6.16.172.in–addr.arpa. Each name is assigned a correspondingly numbered canonical name by its CNAME record in the new 1–63 subdomain. Thus, 5.6.16.172.in–addr.arpa. is assigned the canonical name 5.1–63.6.16.172.in–addr.arpa.. The new 1–63 subdomain is then delegated to arctic and falcon. If a query comes in for 2.6.16.172.in–addr.arpa, the resolver is told that the real name it is seeking is 2.1–63.6.16.172.in–addr.arpa, and that the servers for that name are arctic and falcon. This technique allows you to get around the limitation of delegating reverse subdomains

on a byte boundary.

The reverse zone may seem like a lot of trouble for a little gain; after all, most of the action happens in the hostname space. But keeping the reverse zone up−to−date is important. Several programs use the reverse domain to map IP addresses to names for status displays. netstat is a good example. Some remote systems use reverse lookup to check on who is using a service, and in extreme cases won't allow you to use the service if they can't find your system in the reverse domain. Keeping the reverse domain updated ensures smooth operation.

Running *named*

named is started at boot time by one of the startup scripts. On a Red Hat system, it is started by the /etc/rc.d/init.d/named script. The script checks that the named program and the named.conf file are available, and it then starts named. After the configuration files are created, named will restart whenever the system reboots.

Of course, it is not necessary to reboot in order to run named. You can run the named boot script from the command line. The Red Hat script accepts several arguments:

start Starts named if it is not already running.

stop Terminates the currently running named process.

restart Unconditionally terminates the running named process, and starts a new named process.

condrestart Does the same thing as restart, but only if named is currently running. If named is not running, no action is taken.

reload Uses the named management tool (rndc) to reload the DNS database files into the server. If rndc fails to reload the files, the script attempts to force a reload by sending a signal to named. (More on rndc and named signal processing later in this chapter.)

probe Uses the named management tool (rndc) to reload the DNS database files into the server. If rndc fails to reload the files, the script attempts to start named.

status Displays information about whether or not named is running.

Running boot scripts from the command line is so useful and popular that Red Hat provides a script named /sbin/service for the sole purpose of running boot scripts. The syntax of the service command is simple:

service *script command*

where *script* is the name of a script in the /etc/init.d directory, and *command* is an argument passed to that script.

Here is an example of using the Red Hat startup script to check the status of named and then to start named running:

```
[root]# service named status
named not running.
[root]# service named start
Starting named: [  OK  ]
```

If your Linux system does not have a named script such as the one provided by Red Hat, you can start named from the command line by typing **named &**. However, named is rarely started from the command line because it automatically starts at every boot, and because rndc and signal processing mean that it does not need to be stopped and started to load a new configuration. Before discussing rndc, let's look at how signals can be used to cause named to load a new configuration and to perform a number of other tasks.

named Signal Processing

Signal processing is one area in which the version of BIND matters. BIND 8 and earlier versions of BIND handle several different signals. BIND 9 handles only two: SIGHUP and SIGTERM. Under BIND 9, SIGHUP reloads the DNS database, and SIGTERM terminates the named process. You can use signals with BIND 8, but don't use signals with BIND 9. Control BIND 9 with rndc, which is covered in the next section. That said, BIND 8 accepts the following signals.

The SIGHUP signal causes named to reread the named.conf file and reload the name server database. Using SIGHUP causes the reload to occur immediately. On a master server, this means that the local database files are reloaded into memory. On a slave server, this means that the slave immediately reloads its local disk copies and then sends a query to the master server for the SOA record to check if there is a new configuration.

SIGINT causes named to dump its cache to named_dump.db. The dump file contains all of the domain information that the local name server knows. Examine this file. You'll see a complete picture of the information the server has learned. Examining the cache is an interesting exercise for anyone who is new to DNS.

Use SIGUSR1 to turn on tracing. Each subsequent SIGUSR1 signal increases the level of tracing. Trace information is written to named.run. Tracing can also be enabled with the –d option on the named command line if the problem you are looking for occurs so early in the startup that the SIGUSR1 signal is not useful. The advantage of SIGUSR1 is that it allows tracing to be turned on when a problem is suspected, without stopping and restarting named.

The opposite of SIGUSR1 is SIGUSR2. It turns off tracing and closes the trace file. After issuing SIGUSR2, you can examine the file or remove it if it is getting too large.

The kill command is used to send a signal to a running process. As the name implies, by default it sends the kill signal. To use it to send a different signal, specify the signal on the command line. For example, specify –INT to send the SIGINT signal. The process ID (PID) must be provided on the kill command line to ensure that the signal is sent to the correct process.

You can learn the process ID by using the ps command or using the status argument with the /etc/init.d/named script. For example:

```
$ ps ax | grep named
   271  ?  S    0:00 /usr/sbin/named
  7138  p0 S    0:00 grep named
```

In the case of named, you can learn the process ID by listing the named.pid file:

```
$ cat /var/run/named/named.pid
271
```

Combining some of these commands, you can send a signal directly to named. For example, to reload the name server, you could enter the following command:

```
kill -HUP 'cat /var/run/named/named.pid'
```

The cat /var/run/named/named.pid command that is enclosed in single quotes is processed by the shell first. On our sample system, this returns the PID 271. That is combined with the kill command and then is processed as kill –HUP 271. This works, but it is easier to use the named management tools that come with BIND.

The *named* Control Tools

There are two versions of the named management tool—one for BIND 8 and another one for BIND 9. BIND 8 uses the Name Daemon Control (ndc) tool, and BIND 9 uses the Remote Named Daemon Control (rndc) tool. The commands used with these tools are very similar. When there are differences, the text points them out. Otherwise, rndc commands will work for ndc simply by replacing rndc with ndc on the command line.

The named management tool allows you to control named with much less fuss than sending signals. You don't need to know the correct PID, and you don't need to remember the correct signal. For example, in the previous section, kill was used with the SIGHUP signal to reload the name server. To do the same thing with rndc, you enter **rndc reload**. This command is simple and much more intuitive than the kill command. The valid rndc command–line arguments are listed in Table 4.3. Note that a few commands in Table 4.3 are available only for ndc, and one command is available only for rndc.

Table 4.3: *rndc* Commands

Argument	Function
status	Displays the status of named (ndc only)
dumpdb	Dumps the cache to named_dump.db
reload	Reloads the name server
stats	Dumps statistics to named.stats (ndc only)
trace	Turns on tracing to named.run (ndc only)
notrace	Turns off tracing and closes named.run (ndc only)
querylog	Toggles query logging, which logs each incoming query to syslog
start	Starts named (ndc only)
halt	Stops named without saving pending dynamic updates (rndc only)
stop	Stops named

The biggest difference between ndc and rndc is not the command set. The biggest difference is that rndc allows remote access. To use ndc on a BIND 8 name server, you must log directly into the name server and issue the commands there. rndc can be used from a remote system. The commands sent from the remote system must be cryptographically signed using the HMAC–MD5 algorithm and the correct key. The parameters needed to verify the remote system are defined in the /etc/rndc.conf file.

The rndc.conf file and all the commands it contains apply only to BIND 9—BIND 8 does not use rndc. Listing 4.13 shows the rndc.conf file that is delivered with Red Hat 7.2. Comments have been deleted from the start of the file, but otherwise Listing 4.13 shows the file exactly as it is delivered.

Listing 4.13: The Red Hat *rndc.conf* File

```
/*
 * Sample rndc configuration file.
 */

options {
        default-server localhost;
        default-key     "key";
};

server localhost {
        key     "key";
};

key "key" {
        algorithm       hmac-md5;
        secret "eabDFqxVnhWyhUwoSVjthOue0bYtvQUCiSuBqHxDRWilSaWMoMORNLmyEbJr";
};
```

The rndc.conf file is structured like that of the named.conf file, but rndc.conf can contain, at most, only three different statements. They are

options Like the options statement in the named.conf file, this statement defines options that apply to the entire rndc configuration. But the rndc.conf options statement can contain only two options:

default-server Identifies the name server to which rndc commands are sent if no name server is specified with the –s argument on the rndc command line.

default-key Identifies the algorithm/key pair used if no key–id is specified with the –y argument on the rndc command line.

server The rndc.conf server statement defines the characteristics of a server that accepts rndc commands. Only the key option can be used with the server statement in an rndc.conf file. The key option identifies the key used by the server. The key–id used on the key option must match a key defined in the rndc.conf file.

key The syntax and purpose of the rndc.conf key statement are identical to those of the key statement found in the named.conf file. The key statement assigns a key–id to an algorithm/key pair.

The rndc.conf file is the client side of the rndc configuration. The key associated with a server in the rndc.conf file must be the same key that the server has defined in its named.conf file. Listing 4.13 is part of the configuration delivered by Red Hat for a caching–only server. The key defined in the rndc.conf file is associated with the local host. Therefore, this key should match the key defined in the named.conf file of the local host. Refer to Listing 4.5, which is the named.conf file delivered with the Red Hat caching–only configuration. You will notice that the key defined in Listing 4.5 matches the key defined in Listing 4.13. The keys must match for the client and server to successfully

communicate.

In addition to configuring the client side of rndc in the rndc.conf file, the server side must be configured in the named.conf file. Part of the configuration has already been done. The key statement is already defined in the named.conf file that comes from Red Hat, as shown in Listing 4.5. The other thing that is needed before rndc will work is a properly configured controls statement. Attempting to control the named process with rndc before placing the correct controls statement in the server's named.conf file returns an error, as in this example:

```
[root]# rndc reload
rndc: connect: connection refused
```

The controls statement defines

- the interface and port on which rndc commands are accepted
- the clients allowed to submit rndc commands
- the algorithm/key pair that must be used by the clients to sign the commands

Adding the following controls statement to the configuration shown in Listing 4.5 enables rndc on a server running Red Hat's default caching–only configuration:

```
// a control channel for rndc
controls {
         inet 127.0.0.1 allow { localhost; } keys { "key"; };
};
```

The inet option defines the address of the network interface on which the server will accept rndc commands. In this example, it is the address of the loopback interface, meaning that the server will only accept rndc commands on the internal interface; commands will not be accepted from the network. Optionally, the inet option can be followed by a port option to change the standard port number; for example, inet 127.0.0.1 port 2020. By default, rndc uses port number 953. If a non–standard port is used, the same port number must be specified on the client's rndc command line using the –p argument.

The allow option defines the clients that are permitted to control the server through rndc commands. Clients can be identified by hostname or IP address. In the example, the localhost is granted permission to use rndc. Without this setting, rndc commands entered at the server's console are rejected.

Finally, the keys option is used to identify the algorithm/key pair used to sign rndc transactions. The key–id provided to this option must match a key–id defined in the client's rndc.conf file and elsewhere in the server's named.conf file.

After this controls statement is added to the named.conf file from Listing 4.5, the server is restarted so that it will use the new configuration:

```
[root]# service named restart
Stopping named:                                            [  OK  ]
Starting named:                                            [  OK  ]
[root]# rndc reload
rndc: reload command successful
```

After restarting the server with the new configuration, the rndc reload command that was rejected earlier now runs successfully.

These steps are needed to enable rndc on BIND 9 systems because rndc requires a specific network configuration. Special care must be taken before enabling such a powerful feature that can potentially be accessed via the network. On the other hand, the ndc command that runs under BIND 8 can be used from the server console without any special configuration. ndc also has a controls statement, but because ndc lacks support for strong authentication, it is not used remotely. ndc defaults to local operation, and is ready to run from the local console.

Using the Host Table with DNS

You should always use DNS. But even though you will be using DNS, you will have a host table. Which source of information should your system check first, DNS or the host table?

I usually configure my systems to use DNS first, and to fall back to the host table only when DNS is not running. Your needs may be different. You may have special host aliases that are not included in the DNS database, or local systems that are known only to a small number of computers on your network and therefore are not registered in the official domain. In these cases, you want to check the host table before sending an unanswerable query to the DNS server.

There are two files involved in configuring the order in which name services are queried for information. The host.conf file is used primarily for name service. The nsswitch.conf file covers a wider range of administrative databases, including name service.

The *host.conf* File

The host.conf file defines several options that control how the /etc/hosts file is processed, and how it interacts with DNS. Listing 4.14 illustrates this with a sample host.conf file that contains every possible option.

Listing 4.14: A Complete *host.conf* File

```
# Define the order in which services are queried
order bind hosts nis
# Permit multiple addresses per host
multi on
# Sort addressees to prefer local addresses
reorder on
# Verify reverse domain lookups
nospoof on
# Log "spoof" attempts
spoofalert on
# Remove the local domain for host table lookups
trim foobirds.org
```

The order option defines the order in which the various name services are queried for a hostname or an IP address. The three values shown in the example are the only three values available:

- bind stands for DNS. (As noted earlier, BIND is the name of the software package that implements DNS on Linux systems.)
- hosts stands for the /etc/hosts file.
- nis stands for the Network Information Service (NIS), which is a name service created by Sun Microsystems.

These services are tried in the order they are listed. Given the order command shown in Listing 4.14, we try DNS first, then the /etc/hosts file, and finally NIS. The search stops as soon as a service answers the query.

The multi option determines whether or not multiple addresses can be assigned to the same hostname in the /etc/hosts file. This option is enabled when on is specified, and is disabled when off is specified. You may be wondering why you would want to do this. Well, assume that you have a single computer directly connected to a few different networks—this is called a *multihomed host*. Each network requires an interface, and each interface requires a different IP address. Thus, you have one host with multiple addresses. But also assume that this is your web server, and that you want everyone to refer to it by the hostname www, regardless of the network they connect in from. In this case, you have one hostname associated with multiple addresses, which is just what the multi option was designed for. multi affects only host table lookups; it has no effect on DNS. DNS inherently supports multiple addresses.

When multiple addresses are returned for a name, they are usually used in the order given. Setting the reorder option to on tells the resolver to sort the addresses so that addresses from the local network are preferred. This duplicates the function of the sortlist command found in the resolv.conf file, but it does not have the power and flexibility of the sortlist command.

As you've seen, Domain Name System permits you to look up a hostname and get an address as well as to look up an address and get a hostname; names to addresses are in one database, and addresses to names are in another database. The nospoof option says that the values returned from both databases must match, or your system will reject the hostname and return an error. For example, if the name wren.foobirds.org returns the address 172.16.5.1, but a lookup for the address 172.16.5.1 returns the hostname host0501.foobirds.org, your system will reject the host as invalid. The keyword on enables the feature, and off disables it.

The spoofalert option is related to the nospoof option. When spoofalert is turned on, the system logs any of the hostname/address mismatches described previously. When spoofalert is turned off, these events are not logged.

The trim option removes the specified domain name from hostnames retrieved from DNS. Given the trim command in Listing 4.14, the hostname hawk.foobirds.org is trimmed to hawk. Multiple trim commands can be included in the host.conf file.

Real host.conf files don't actually use all of these commands. The host.conf file that comes with Red Hat 7.2 has only one line:

```
$ cat /etc/host.conf
order hosts,bind
```

The real heart of the host.conf file is the order command, which defines the order in which the name services are searched. Another file, nsswitch.conf, is also used to define the order in which name services are used, along with the ordering for many other system administration databases.

The *nsswitch.conf* File

The nsswitch.conf file handles much more than just the order of precedence between the host table and DNS. It defines the sources for several different system administration databases because it is an outgrowth of the Network Information System (NIS). NIS makes it possible to centrally control and distribute a wide range of system administration files. Table 4.4 lists all of the administrative databases controlled by the nsswitch.conf file. Unless you run NIS on your network, the sources of

all of these administrative databases, except for the hosts database, will probably be the local files.

Table 4.4: Databases Controlled by *nsswitch.conf*

Database	Holds
aliases	E−mail aliases
ethers	Ethernet addresses for Reverse ARP (RARP)
group	Group Ids
hosts	Hostnames and IP addresses
netgroup	Network groups for NIS
network	Network names and numbers
passwd	User account information
protocols	IP protocol numbers
publickey	Keys for secure RPC (remote procedure call)
rpc	RPC names and numbers
services	Network service port numbers
shadow	User passwords

The hosts entry is the one we are interested in because it indicates the source for hostname and IP address information. In the following sample nsswitch.conf file, DNS is used as the primary source with the local file as the backup source. If DNS can successfully answer the query, it's finished. If DNS can't answer the query, the resolver tries the local file, which in this case is / etc/hosts. Listing 4.15 shows an nsswitch.conf file for a system that does not run NIS.

Listing 4.15: A Sample *nsswitch.conf* File

```
# Sample for system that does not use NIS

passwd:     files
shadow:     files
group:      files

hosts:      dns files
aliases:    files

services:   files
networks:   files
protocols:  files
rpc:        files
ethers:     files
netgroup:   files
publickey:  files
```

Listing 4.15 shows that each database is listed, along with the source for that database. In this example, only the hosts entry has more than one source—first DNS (dns) and then the local hosts file (files). To check the host table before DNS, simply reverse the order:

hosts: files dns

All of the other entries in the sample nsswitch.conf file point to local files as the source of information for those databases. You're already familiar with several of the local files: /etc/passwd,

/etc/group, /etc/shadow, /etc/services, /etc/protocols, /etc/networks, and /etc/hosts. Local files are used for all of these databases unless you run NIS.

In addition to dns and files, there are some other possible source values. nis and nisplus are valid source values if you run NIS or NIS+ on your network. db is used if the local file is a structured database instead of a flat file. hesiod is used if the source of information is a Hesiod server. ldap is used if the source of information is an LDAP server. Also, compat is a valid source field value that might be of use if you run NIS. compat means that the source is a local file, but the local files should be read in a way that is compatible with the old SunOS 4.*x* system. Under SunOS 4.*x*, NIS data could be appended to a file by using a plus sign (+) as the last entry in a file. For example, if /etc/passwd ended with a +, the system would use the accounts in the password file plus every account in NIS. SunOS 4.x has been out of production for several years, and the nsswitch.conf file supercedes the old "plus syntax." However, some people still use it, and the compat function is there if you need it.

In Sum

Name service is a fundamental service of a Linux network. A name service converts text–based hostnames to the numeric IP address required by the network. In the same way that the services in Chapter 3 needed user IDs (UIDs) and group IDs (GIDs) to identify users, networks need IP addresses to identify computers. Domain Names System (DNS) is the tool that maps hostnames to IP addresses for the network.

DNS is implemented on most Linux systems with the Berkeley Internet Name Domain (BIND) software, which is the most widely used DNS software in the Internet. Linux is a fairly new operating system, but it benefits from the fact that it can run venerable software packages such as BIND that have a very long history with many years of debugging and refinement. Linux developers wisely used these tried–and–true packages for the most critical network servers.

E–mail is another critical service that must be provided by every modern network, and most Linux systems use sendmail, which is the most widely used SMTP mail server software, to provide e–mail service. In the next chapter, we configure an e–mail server using sendmail.

Chapter 5: Configuring a Mail Server

Overview

Electronic mail is still the most important user service on the network. The Web carries a greater volume of traffic, but e−mail is the service used for most person−to−person communication. And person−to−person communication is the real foundation of business. No network is complete without e−mail, and no network server operating system is worth its salt if it doesn't include full TCP/IP mail support.

Simple Mail Transport Protocol (SMTP) is the TCP/IP mail transport protocol. Linux provides full SMTP support through the sendmail program, although sendmail does more than just send and receive SMTP mail. sendmail provides mail aliases and acts as a "mail router," routing mail from all of the different user mail programs to the various mail delivery programs while ensuring that the mail is properly formatted for delivery.

This chapter looks at your role in configuring each of these functions. Configuring sendmail can be a large and complex task, but it doesn't have to be. Compared to some network server systems that require a second installation just to install the SMTP server software, Linux distributions do a lot of the configuration for you, and for most sites, the default configuration works fine. This chapter will give you the information you need to make intelligent decisions about when and how to change the default configuration.

sendmail configurations are built using the m4 macro processing language. The output of the m4 process is the sendmail.cf file, which is the configuration file read by sendmail. To fully understand and manage sendmail, you need to understand its functions, the sendmail.cf file from which it reads its configuration, and the m4 macros used to build that file. This chapter covers all three topics.

Using Mail Aliases

Mail aliases are defined in the aliases file. The location of the aliases file is set in the sendmail configuration file. (You'll see this configuration file later in the chapter.) On Linux systems, the file is usually located in the /etc directory (/etc/aliases), and it is occasionally located in the /etc/ mail directory. The basic format of entries in the file is

```
alias: recipient
```

The *alias* is the username in the e−mail address, and *recipient* is the name to which the mail should be delivered. The *recipient* field can contain a username, another alias, or a final delivery address. Additionally, there can be multiple recipients for a single alias.

sendmail aliases perform important functions that are an essential part of creating a mail server. Mail aliases do the following:

> **Specify nicknames for individual users** Nicknames can be used to direct mail addressed to special names, such as postmaster or root, to the real users that do those jobs. When used in conjunction with the domain MX records covered in Chapter 4, "Linux Name Services," aliases can be used to create a standard e−mail address structure for a domain.

Forward mail to other hosts sendmail aliases automatically forward mail to the host address included as part of the recipient address.

Define mailing lists An alias with multiple recipients is a mailing list.

Listing 5.1 is the aliases file that comes with a Red Hat system, with a few additions to illustrate all of these uses.

Listing 5.1: A Sample *aliases* File

```
#
#       @(#)aliases     8.2 (Berkeley) 3/5/94
#
#  Aliases in this file will NOT be expanded in the header from
#  Mail, but WILL be visible over networks or from /bin/mail.
#
#       >>>>>>>>>>      The program "newaliases" must be run after
#       >> NOTE >>      this file is updated for any changes to
#       >>>>>>>>>>      show through to sendmail.
#

# Basic system aliases -- these MUST be present.
mailer-daemon:  postmaster
postmaster:     root


# General redirections for pseudo accounts.
bin:            root
daemon:         root
adm:            root
lp:             root
sync:           root
shutdown:       root
halt:           root
mail:           root
news:           root
uucp:           root
operator:       root
games:          root
gopher:         root
ftp:            root
nobody:         root
apache:         root
named:          root
xfs:            root
gdm:            root
mailnull:       root
postgres:       root
squid:          root
rpcuser:        root
rpc:            root

ingres:         root
system:         root
toor:           root
manager:        root
dumper:         root
abuse:          root

newsadm:        news
newsadmin:      news
```

```
usenet:         news
ftpadm:         ftp
ftpadmin:       ftp
ftp-adm:        ftp
ftp-admin:      ftp
webmaster:      root


# trap decode to catch security attacks
decode:         root

# Person who should get root's mail
root:          staff

# System administrator mailing list
staff: kathy, craig, david@parrot, sara@hawk, becky@parrot
owner-staff: staff-request
staff-request: craig

# User aliases
norman.edwards:  norm
edwardsn: norm
norm: norm@hawk.foobirds.org
rebecca.hunt: becky@parrot
andy.wright: andy@falcon.foobirds.org
sara.henson: sara@hawk
kathy.McCafferty: kathy
kathleen.McCafferty: kathy
```

The Red Hat /etc/aliases file opens with several comment lines. Ignore the information about which mail programs display aliases in the headers of mail messages; it is not really significant. The comment that is significant is the one that tells you to run newaliases every time you update this file. sendmail does not read the /etc/aliases file directly. Instead, it reads a database file produced from this file by the newaliases command.

The first 40 or so lines define aliases for special names. All of them, except the webmaster alias that we added, come preconfigured in the Red Hat aliases file. The first two are aliases that people expect to find on any system running sendmail. Most of the others are aliases assigned to the daemon usernames that are found in the /etc/passwd file. No one can actually log in using the daemon usernames, so any mail that might be directed to these pseudo accounts is forwarded to a real user account. In Listing 5.1, all of this mail is forwarded to the root user account—even mail addressed to newsadm and ftpadm, which at first glance appears to be routed to accounts other than root. For example, newsadm appears to be routed to the user account news, but closer examination reveals that news is itself an alias that is routed to the root account. Aliases can point to other aliases, but eventually they must resolve to a real e–mail account for mail to be successfully delivered.

Of course, you don't really want people logging in to the root account just to read mail, so the aliases file also has an alias for root. In the example, we edited the root entry to forward all mail addressed to root to staff, which is another alias. Notice how often aliases point to other aliases. Doing this is very useful because it allows you to update one alias instead of many when the real user account that the mail is delivered to changes.

The staff alias is a mailing list. A mailing list is simply an alias with multiple recipients. In the example, several people are responsible for maintaining this mail server. Messages addressed to root are delivered to all of these people through the staff mailing list.

Two special aliases are associated with the mailing list. The owner–staff alias is a special alias used by sendmail for error messages relating to the staff mailing list. The format that sendmail requires for this special alias is owner–*list*, where *list* is the name of the mailing list. The other special alias, staff–request, is not required by sendmail, but it is expected by remote users. By convention, manual mailing list maintenance requests, such as being added to or deleted from a list, are sent to the alias *list*–request, where *list* is the name of the mailing list.

The last eight lines are user aliases we added to the file. These lines direct mail received at the mail server to the computers where the users read their mail. These aliases can be in a variety of formats to handle the various ways that e–mail is addressed to a user. The first three lines that forward mail to norm@hawk.foobirds.org all illustrate this. Assume that this /etc/aliases file is on wren, and that the MX record in DNS says that wren is the mail exchanger for foobirds.org. Then, mail addressed to norman.edwards@foobirds.org would actually be delivered to norm@hawk.foobirds.org. It is the combination of mail aliases and MX records that make possible the simplified mail–addressing schemes used at so many organizations.

Defining Personal Mail Aliases

As the last eight lines in the Red Hat aliases file illustrate, one of the main functions of the alias file is to forward mail to other accounts or other computers. The aliases file defines mail forwarding for the entire system. The .forward file, which can be created in any user's home directory, defines mail forwarding for an individual user.

It is possible to use the .forward file to do something that can be done in the /etc/aliases file. For example, if Norman Edwards had an account on a system, but didn't really want to read his mail on that system, he could create a .forward file in his home directory with the following entry:

```
norm@hawk.foobirds.org
```

This entry forwards all mail received in his account on the local system to the norm account at hawk.foobirds.org. However, if you want to permanently forward mail to another account, create an alias in the /etc/aliases file. Simple forwarding is not the primary use for the .forward file. A much more common use for the file is to invoke special mail processing before mail is delivered to your personal mail account.

Using *sendmail* to Receive Mail

sendmail runs in two different ways. When you send mail, a sendmail process starts, delivers your mail, and then terminates. To receive mail, sendmail runs as a persistent daemon process. The –bd option tells sendmail to run as a daemon and to listen to TCP port 25 for incoming mail. Use this option to accept incoming TCP/IP mail. Without it, your system will not collect inbound mail. As you'll see in Chapter 11, "More Mail Service," many systems do not collect inbound SMTP mail. Instead, they use protocols such as POP and IMAP to move mail from the mailbox server to the mail reader. In general, however, most Linux systems are configured to run sendmail as a daemon. The code that runs the sendmail daemon in the Slackware Linux /etc/rc.d/ rc.M startup script is very straightforward:

```
/usr/sbin/sendmail -bd -q 15m
```

The code runs sendmail with the –bd and –q options. In addition to listening for inbound mail, the sendmail daemon periodically checks to see if there is mail waiting to be delivered. It's possible that

a sendmail process that was started to send a message was not able to successfully deliver the mail. In that case, the process writes the message to the mail queue, and counts on the daemon to deliver it at a later time. The –q option tells the sendmail daemon how often to check the undelivered mail queue. In the Slackware example, the queue is processed every 15 minutes (–q 15m).

The code that Red Hat uses to start the sendmail daemon is found in the /etc/rc.d/init.d/sendmail script. It is more complex than the code used by Slackware because Red Hat uses script variables read from an external file to set the command–line options. The file it reads is /etc/sysconfig/sendmail, which normally contains these two lines:

```
DAEMON=yes
QUEUE=1h
```

If the variable DAEMON is equal to yes, sendmail is started with the –bd option. The QUEUE variable sets the time value of the –q option. In this case, it is one hour (1h), which is a value that I like even more than the 15 minutes used by Slackware. Don't set the –q value too low. Processing the queue too often can cause problems if the queue grows very large due to a delivery problem such as a network outage. To change the queue value on a Red Hat system, edit the /etc/sysconfig/sendmail file.

The *sendmail* Configuration File

The file that defines the sendmail runtime configuration is sendmail.cf, which is a large, complex file that is divided into seven different sections. The file is so large and so complex that system administrators are often intimidated by it. You needn't be. The file is designed to be easily parsed by sendmail, not to be easily written by a system administrator. But normally, you don't directly write to this file. Instead, you build the file with the m4 commands described later in this chapter. It is important to have a basic understanding of the syntax and structure of the sendmail.cf file in order to better understand the effect of the m4 commands and to gain the mastery needed for troubleshooting. Yet it is equally important to realize that you don't have to build the sendmail.cf file by hand.

The section labels from the Red Hat sendmail.cf file provide an overview of the structure and the function of the file. The sections, each examined in detail in this chapter, are as follows:

Local Info This section defines the configuration information specific to the local host.

Options This section sets the options that define the sendmail environment.

Message Precedence This section defines the sendmail message precedence values.

Trusted Users This section defines the users who are allowed to change the sender address when they are sending mail.

Format of Headers This section defines the headers that sendmail inserts into mail.

Rewriting Rules This section holds the commands that rewrite e–mail addresses from user mail programs into the form required by the mail–delivery programs.

Mailer Definitions This section defines the programs used to deliver the mail. The rewrite rules used by the mailers are also defined in this section.

Note All Linux sendmail.cf files have the same structure because they are all created from the m4 macros (covered later in this chapter and in Appendix C, "The M4 Macros for Sendmail") that come in the sendmail distribution.

The Local Info Section

Local Info, the first section in the sendmail.cf file, contains the hostname, the names of any mail relay hosts, and the mail domain. It also contains the name that sendmail uses to identify itself when it returns error messages, as well as the version number of the sendmail.cf file.

The local information is defined by D commands that define macros, C commands that define class values, F commands that load class values from files, and K commands that define databases of information. Some sample lines lifted from the Local Info section of the Red Hat sendmail.cf file are shown in Listing 5.2. The commands have been reordered, and a comment has been added to make the commands more understandable, but the commands themselves are just as they appear in the original file.

Listing 5.2: Sample of the *sendmail.cf* Local Info Section

```
# my name for error messages
DnMAILER-DAEMON

# operators that cannot be in local usernames
CO @ % !

# host name aliases for this system
Cwlocalhost
# file containing names of hosts for which we receive email
Fw/etc/mail/local-host-names

# Access list database (for spam stomping)
Kaccess hash -o /etc/mail/access.db
```

Lines that begin with # are comments. The first real command in the sample is a define macro (D) command that defines the username that sendmail uses when sending error messages. The macro being defined is n. Many macro names are only a single upper– or lowercase character. When a long name is used, the name is enclosed in curly braces, for example, {verify}.

The value assigned to n is MAILER–DAEMON. After a value is stored in a macro, it can be recalled later in the configuration using the syntax $*x*, where *x* is the name of the macro. Thus, commands later in the configuration that need to send error messages can use $n to retrieve the correct sender name. Setting a macro value once at the beginning of the configuration affects commands throughout the configuration, which simplifies customization.

The first class command (C) assigns the values @, %, and ! to the class variable O. These three values are characters that cannot be used in local usernames because they would screw up e–mail. A class is an array of values. Classes are used in pattern–matching to check whether or not a values matches one of the values in a class, using the syntax $=*x*, where *x* is the name of the class. A command containing the string $=O is testing a value to see if it is equal to @, % or !.

The second C command stores the string localhost into the class variable w, which holds a list of

valid hostnames for which the local computer will accept mail. Normally, if a system running sendmail receives mail addressed to another hostname, it assumes that the mail belongs to that host. If your system should accept the mail, even if it appears to be addressed to another host, the name of that other host should be stored in class w. Listing 5.2 stores only one value in w. You could add additional hostnames, separated by spaces, directly to the C command line, but there is an easier way to add values to a class variable.

The file command (F) adds the values found in the file /etc/mail/local–host–names to the class w variable. F is the command, w is the name of the class variable, and /etc/mail/local–host–names is the path of the file that is to be stored in the variable. External files and databases make it possible to control sendmail's behavior without directly modifying the sendmail configuration. Flat files such as local–host–names are only part of the story. sendmail also uses structured database files.

The last command in Listing 5.2 defines an e–mail address database. The K command declares a database named "access." The database is in the hash format, which is a standard Unix database format. The file that contains the database is /etc/mail/access.db. All of this information (the internal name, the database type, and the file that holds the database) is defined by the K command. Subsequent commands in the sendmail.cf file use the database to match patterns, to retrieve values, and to perform security checks. How databases are used is covered in detail in Chapter 11, when the access database is used to control mail relaying and delivery.

These four types of commands illustrate everything that is done in the Local Info section of the sendmail.cf file. This section is the most important section of the file from the standpoint of a system administrator trying to understand a configuration because it is the part of the sendmail.cf file that stores the variables used to customize the configuration.

The Options Section

Options define the sendmail environment. All of the option values are used directly by the sendmail program. There are nearly 100 options, but a few samples from the Red Hat sendmail.cf file can illustrate what options do.

Listing 5.3: Sample *sendmail.cf* Options

```
# location of alias file
O AliasFile=/etc/aliases
# Forward file search path
O ForwardPath=$z/.forward.$w:$z/.forward
# timeouts (many of these)
O Timeout.queuereturn=5d
O Timeout.queuewarn=4h
```

These options all have something to do with sendmail functions that have already been discussed. The first option command (O) sets the location of the aliases file to /etc/aliases. The second option defines the location of the .forward file. Notice the $z and $w included in this option. These are macro values. The $w macro contains the computer's hostname, indicating that it is possible to use the computer's hostname as a filename extension on a .forward file. Given the fact that you already know that the .forward file is found in the user's home directory, you can guess that the value of the $z macro is the user's home directory.

The last two options in the example relate to processing the queue of undelivered mail. The first of these options tells sendmail that if a piece of mail stays in the queue for five days (5d), it should be

returned to the sender as undeliverable. The second of these options tells sendmail to send the user a warning message if a piece of mail has been undeliverable for four hours (4h). Many of the options in the sendmail.cf file set the timer values used by sendmail.

File locations and other things that vary based on the operating system being used are handled in the options section. The m4 macros build a sendmail.cf customized for the target operating system. Because of this, the options in the sendmail.cf file that comes with your Linux system are probably correct for that system.

The Message Precedence Section

Message Precedence is used to assign priority to messages entering the queue. By default, mail is considered "first–class mail," and is given a precedence of 0. The higher the precedence number, the higher the priority of the message.

But don't get excited. Increasing priority is essentially meaningless. About the only useful thing you can do is to select a negative precedence number, which indicates low–priority mail. Because error messages are not generated for mail with a negative precedence number, low priorities are useful for mass mailings. The precedence values from the Red Hat sendmail.cf are

```
Pfirst-class=0
Pspecial-delivery=100
Plist=-30
Pbulk=-60
Pjunk=-100
```

Precedence values have very little importance. To request a precedence, mail must include a Precedence header, which it very rarely does. The five precedence values included in the sendmail.cf file that comes with your Linux system are more than you'll ever need.

The Trusted Users Section

Trusted users are allowed to change the sender address when they are sending mail. Trusted users must be valid usernames from the /etc/passwd file. The trusted users defined in the sendmail.cf file that comes with your Linux system are root, uucp, and daemon:

```
Troot
Tdaemon
Tuucp
```

The T commands define trusted users. The list of trusted users is stored in class t. Thus the three previously listed T commands could be replaced by three C commands:

```
Ctroot
Ctdaemon
Ctuucp
```

Likewise, an F command can be used to load class t from a file. The "Trusted users" section of the sendmail.cf file on our sample Red Hat Linux 7.2 system does exactly that with the following command:

```
Ft/etc/mail/trusted-users
```

A quick check of /etc/mail/trusted–users shows that the file is empty. Therefore, the Red Hat system uses only the three trusted users (root, daemon and uucp) used on most system. To add trusted users to our sample system, place the usernames in the trusted–users file.

Do not modify the Trusted Users list without a very good reason. Adding users to this list is a potential security problem.

The Format of Headers Section

Mail headers are those lines found at the beginning of a mail message that provide administrative information about the mail message, such as when and from where it was sent. The Format of Headers section defines the headers that sendmail inserts into mail. The header definitions from the Red Hat sendmail.cf file are shown in Listing 5.4.

Listing 5.4: *sendmail.cf* Header Commands

```
H?P?Return-Path: <$g>
HReceived: $?sfrom $s $.$?_($?s$|from $.$_)
        $.$?{auth_type}(authenticated$?{auth_ssf} ($${auth_ssf} bits)$.)
        $.by $j ($v/$Z)$?r with $r$. id $i$?{tls_version}
        (using ${tls_version} with cipher ${cipher} (${cipher_bits} bits)
 verified ${verify})$.$?u
        for $u; $|;
        $.$b
H?D?Resent-Date: $a
H?D?Date: $a
H?F?Resent-From: $?x$x <$g>$|$g$.
H?F?From: $?x$x <$g>$|$g$.
H?x?Full-Name: $x
H?M?Resent-Message-Id: <$t.$i@$j>
H?M?Message-Id: <$t.$i@$j>
```

Each header line begins with the H command, which is optionally followed by header flags enclosed in question marks. The header flags control whether or not the header is inserted into mail that is bound for a specific mailer. If no flags are specified, the header is used for all mailers. If a flag is specified, the header is used only for a mailer that has the same flag set in the mailer's definition. (Mailer definitions are covered later in this chapter.) Header flags only control header insertion. If a header is received in the input, it is passed to the output, regardless of the flag settings.

Each line also contains a header name, a colon, and a header template. These fields define the structure of the actual header. Macros in the header template are expanded before the header is inserted into a message. Look at the first header in the sample. $g says to use the value stored in the g macro, which holds the sender's e–mail address. Assume the sender is David. After the macro expansion, the header might contain

```
Return-Path: <david@wren.foobirds.com>
```

Note The second header shows examples of long macro names, such as {auth_ type}.
The sample headers provide examples of a conditional syntax that can be used in header templates and macro definitions. It is an if/else construct where $? is the "if," $| is the "else," and $. is the "endif." A simple example from Listing 5.4 is

```
H?F?Resent-From: $?x$x <$g>$|$g$.
```

The header template $?x$x <$g>$|g says that if ($?) macro x exists, use $x <$g> as the header template, else ($|) use $g as the template. Macro x contains the full name of the sender. Thus if it exists, the header is

```
Resent-From: David Craig <david@wren.foobirds.org>
```

If x doesn't exist, the header is

```
Resent-From: david@wren.foobirds.org
```

The headers provided in your system's sendmail.cf file should be correct and sufficient for your installation. Header formats are defined in RFC standards documents. There is no need to create customized headers.

The Rewriting Rules Section

The Rewriting Rules section defines the rules used to parse e−mail addresses from user mail programs and rewrite them into the format required by the mail delivery programs. *Rewrite rules* match the input address against a pattern, and if a match is found, rewrite the address into a new format using the rules defined in the command.

The left side of a rewrite rule contains a pattern defined by macro and literal values and by special symbols. The right side of a rewrite rule defines the template used to rewrite addresses that match the pattern. The template is also defined with literals, macro values, and special symbols. Literals are simply literal string values. Macros are the macro and class values defined in the Local Info section of the sendmail.cf file. The special symbols vary, depending on whether they are used in the left−side pattern or the right−side template. Table 5.1 lists the pattern matching symbols and Table 5.2 lists the template symbols.

Table 5.1: Pattern Matching Symbols

Symbol	Meaning
$@	Match exactly zero tokens.
$*	Match zero or more tokens.
$−	Match exactly one token.
$+	Match one or more tokens.
$*x*	Match all tokens in macro variable *x*.
$=*x*	Match any token in class variable *x*.
$~*x*	Match any token not in class variable *x*.

Table 5.2: Rewrite Template Symbols

Symbol	Purpose
$*n*	Insert the value from indefinite token *n*.
$:	Terminate this rewrite rule.
$@	Terminate the entire ruleset.
$>*name*	Call the ruleset identified as *name*.
$[*hostname*$]	Convert *hostname* to DNS canonical form.
$(*database−spec*$)	Get the value from a database.

Rewrite rules divide e−mail addresses into tokens for processing. A *token* is a string of characters delimited by an operator defined in the OperatorChars option. The operators also count as tokens. Based on this, the address kathy@parrot contains three tokens: the string kathy, the operator @, and the string parrot.

The tokens from the input address are matched against the pattern. The macro and literal values are directly matched against values in the input address, and the special symbols match the remaining tokens. The tokens that match the special symbols in a pattern are identified numerically, according to their relative position in the pattern that they match. Thus, the first group of tokens to match a special symbol is called $1, the second is $2, the third is $3, and so on. These tokens can then be used to create the rewritten address.

An example will clarify how addresses are processed by rewrite rules. Assume the input address is

```
kathy@parrot
```

Assume the current rewrite rule is

```
R$+@$-    $1<@$2.$D>   user@host -> user<@host.domain>
```

The R is the rewrite command, $+@$− is the pattern against which the address is matched, and $1<@$2.$D> is the template used to rewrite the address. The remainder of the command line is a comment that is intended to clarify what the rule does.

The input address matches the pattern because

- It contains one or more tokens before the literal @, which is what the special symbol $+ requires. The token that matches the special symbol is the string kathy. This token can now be referred to as $1 because it matched the first special symbol.
- It contains an @ that matches the pattern's literal @.
- It contains exactly one token after the @, which is what the $− requires. The token that matches this special symbol is the string parrot, which can now be referenced as $2 because it matched the second special symbol.

The template that rewrites the address contains the token $1, a literal string <@, the token $2, a literal dot (.), the value stored in macro D, and the literal >. You know that $1 contains kathy, and $2 contains parrot. Assume that the macro D was defined elsewhere in the sendmail.cf file as foobirds.org. Given these values, the input address is rewritten as

```
kathy<@parrot.foobirds.org>
```

A rewrite rule may process the same address several times because after being rewritten, the address is again compared against the pattern. If it still matches, it is rewritten again. The cycle of pattern matching and rewriting continues until the address no longer matches the pattern. In our example, when the address is again compared to the pattern after rewriting, it fails to match the pattern a second time because it no longer contains exactly one token after the literal @. In fact, it now has six tokens after the @: parrot, ., foobirds, ., org, and >. So no further processing is done by this rewrite rule, and the address is passed to the next rule in line.

Rulesets

Individual rewrite rules are grouped together in rulesets, so that related rewrite rules can be referenced by a single name or number. The S command marks the beginning of a ruleset; and

identifies it with a name, number, or both. Therefore, the command S4 marks the beginning of ruleset 4, SLocal_check_mail marks the beginning of the Local_check_mail ruleset, and Scanonify=3 defines the beginning of the canonify ruleset, which is also known as ruleset 3.

Five rulesets are called directly by sendmail to handle normal mail processing:

- Ruleset canonify, also known as ruleset 3, is called first to prepare all addresses for processing by the other rulesets.
- Ruleset parse, also known as ruleset 0, is applied to the mail delivery address to convert it to the (mailer, host, user) triple, which contains the name of the mailer that will deliver the mail, the recipient hostname, and the recipient username.
- Ruleset sender, also known as ruleset 1, is applied to all sender addresses.
- Ruleset recipient, also known as ruleset 2, is applied to all recipient addresses.
- Ruleset final, also known as ruleset 4, is called last to convert all addresses from internal address formats into external address formats.

There are three basic types of addresses: delivery addresses, sender addresses, and recipient addresses. A recipient address and a delivery address sound like the same thing, but there is a difference. As the mailing list alias illustrated, there can be many recipients for a piece of mail, but mail is delivered to only one person at a time. The recipient address of the one person to which the current piece of mail is being delivered is the delivery address. Different rulesets are used to process the different types of addresses.

Figure 5.1 shows the rulesets that handle each address type. The S and R symbols in Figure 5.1 represent rulesets that have names, just like the other rulesets, but the specific rulesets used are identified by the S and R fields of the mailer definition. Each mailer specifies its own S and R rulesets to process sender and recipient addresses in a manner required by the mailer.

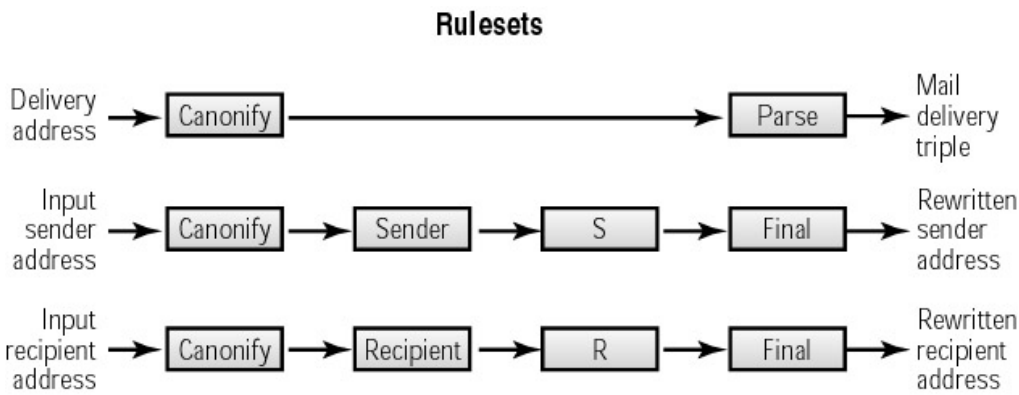


Figure 5.1: sendmail rulesets

The Mailer Definitions Section

The Mailer Definitions section defines the instructions used by sendmail to invoke the mail delivery programs. The specific rewrite rules associated with each individual mailer are also defined in this section. Mailer definitions begin with the mailer command (M). Searching through the Mailer Definitions section of the Red Hat configuration file for lines that begin with M produces the mailer definitions in Listing 5.5.

Listing 5.5: Sample mailer Definitions

```
Mlocal,         P=/usr/bin/procmail, F=lsDFMAw5:/|@qSPfhn9,
                S=EnvFromL/HdrFromL, R=EnvToL/HdrToL,
                T=DNS/RFC822/X-Unix,
```

```
                A=procmail -t -Y -a $h -d $u
Mprog,          P=/usr/sbin/smrsh, F=lsDFMoqeu9,
                S=EnvFromL/HdrFromL, R=EnvToL/HdrToL,
                D=$z:/, T=X-Unix/X-Unix/X-Unix,
                A=smrsh -c $u
Msmtp,          P=[IPC], F=mDFMuX,
                S=EnvFromSMTP/HdrFromSMTP, R=EnvToSMTP,
                E=\r\n, L=990,
                T=DNS/RFC822/SMTP,
                A=TCP $h
Mesmtp,         P=[IPC], F=mDFMuXa,
                S=EnvFromSMTP/HdrFromSMTP, R=EnvToSMTP,
                E=\r\n, L=990,
                T=DNS/RFC822/SMTP,
                A=TCP $h
Msmtp8,         P=[IPC], F=mDFMuX8,
                S=EnvFromSMTP/HdrFromSMTP, R=EnvToSMTP,
                E=\r\n, L=990,
                T=DNS/RFC822/SMTP,
                A=TCP $h
Mdsmtp,         P=[IPC], F=mDFMuXa%,
                S=EnvFromSMTP/HdrFromSMTP, R=EnvToSMTP,
                E=\r\n, L=990,
                T=DNS/RFC822/SMTP,
                A=TCP $h
Mrelay,         P=[IPC], F=mDFMuXa8,
                S=EnvFromSMTP/HdrFromSMTP, R=MasqSMTP,
                E=\r\n, L=2040,
                T=DNS/RFC822/SMTP,
                A=TCP $h
Mprocmail,      P=/usr/bin/procmail, F=DFMSPhnu9,
                S=EnvFromSMTP/HdrFromSMTP,
                R=EnvToSMTP/HdrFromSMTP,
                T=DNS/RFC822/X-Unix,
                A=procmail -Y -m $h $f $u
```

The first two mailer definitions in Listing 5.5 are required by sendmail. The first of these defines a mailer for local mail delivery. This mailer must always be called "local." The P argument defines the path to the local mailer. In this configuration, procmail is used as the local mailer. The second definition is for a mailer that delivers mail to programs, which is always called "prog". The P argument points to a program named smrsh, which is the Sendmail Restricted Shell—a special shell program specifically for handling mail. sendmail expects to find both of these mailers in the configuration, and requires that they be given the names "local" and "prog". All other mailers can be named anything the system administrator wishes. However, in practice, that is not the case. Because the sendmail.cf files on Linux systems are built from the same m4 macros, they use the same mailer names

The next five mailer commands define mailers for TCP/IP mail delivery. The first one, designed to deliver traditional 7–bit ASCII SMTP mail, is called smtp. The next mailer definition, which is for Extended SMTP mail, is called esmtp. The smtp8 mailer definition handles unencoded 8–bit SMTP data. The dsmtp mailer provides support for on–demand SMTP, which is special form of SMTP in which the recipient downloads mail instead of the normal case in which the sender initiates mail transfer. Finally, relay is a mailer that relays TCP/IP mail through an external mail relay host. Of these, only esmtp, which is the default mailer, and relay are actually used anywhere in the basic configuration.

The last mailer definition in Listing 5.5 is for procmail. procmail (covered in Chapter 11) is an

optional mailer found on most Linux systems. The A argument in this definition invokes procmail with the –m command–line argument, which allows procmail to be used for mail filtering. Like most of the SMTP mailers, this mailer is not used anywhere in the basic sendmail.cf file. These unused definitions provide a complete set of mailers, but they are not needed for most configurations.

Examining any one of the mailer entries, such as the entry for the smtp mailer, explains the structure of all of them:

M Beginning a line with an M indicates that the command is a mailer definition.

smtp Immediately following the M is the name of the mailer, which can be anything you wish. In this sample, the name is smtp.

P=[IPC] The P argument defines the path to the program used for this mailer. In this case, it is [IPC], which means this mail is delivered by sendmail. Other mailer definitions, such as local, have the full path of some external program in the field.

F=mDFMuX The F argument defines the sendmail flags for this mailer. Other than knowing that these are mailer flags, the meaning of each individual mailer flag is of little interest because the flags are correctly set by the m4 macro that builds the mailer entry. In this example, m says that this mailer can send to multiple recipients at once; DFM says that Date, From, and Message–ID headers are needed; u says that uppercase should be preserved in hostnames and usernames; and X says that message lines beginning with a dot should have an extra dot prepended.

S=EnvFromSMTP/HdrFromSMTP The S argument defines the S rulesets illustrated in Figure 5.1. The rulesets can be different for every mailer, allowing different mailers to process e–mail addresses differently. In this case, the sender address in the mail "envelope" is processed through ruleset EnvFromSMTP, also known as ruleset 11, and the sender address in the message is processed through ruleset HdrFromSMTP, also known as ruleset 31. (You'll see more on this later when a sendmail configuration is tested.)

R=EnvToSMTP The R argument defines the R ruleset shown in Figure 5.1. This value can be different for every mailer to allow each mailer to handle addresses differently. The sample mailer processes all recipient addresses through ruleset EnvToSMTP, also known as ruleset 21. Only one ruleset is used for the R argument with the smtp mailer; however, it is possible to specify two different rulesets, one for envelope processing and one for header processing, in exactly the same way as two rulesets were defined for the S argument.

E=\r\n The E argument defines how individual lines in a message are terminated. In this case, lines are terminated with a carriage return and a line feed.

L=990 The L argument defines the maximum line length for this mailer. This mailer can handle messages that contain individual lines up to 990 bytes long.

T=DNS/RFC822/SMTP The T argument defines the MIME types for messages handled by this mailer. This mailer uses DNS for hostnames, RFC822 e–mail addresses, and SMTP error codes.

A=TPC $h The A argument defines the command used to execute the mailer. In this

case, the argument refers to an internal sendmail process. (Note that TCP and IPC can be used interchangeably.) In other cases (the local mailer is a good example), the A argument is clearly a command line.

The mailer definitions that come with your Linux system will include local, prog and the SMTP mailers. These are the correct mailer definitions to run sendmail in a TCP/IP network environment.

Configuring the *sendmail.cf* File

It's important to realize how rarely the sendmail.cf file needs to be modified on a typical Linux system. The configuration file that comes with your Linux system will work. Generally, you modify the sendmail configuration not because you need to, but because you want to. You modify it to improve the way things operate, not to get them to operate. To illustrate this, let's look at the default Red Hat configuration on the system parrot.foobirds.org.

Using the default configuration, the From address on outbound e–mail is user@parrot.foobirds.org. This is a valid address, but assume that it's not exactly what you want. In the last chapter, you defined MX records for the domain. To use them, you want people to use addresses in the form *user*@foobirds.org, so you don't want the hostname in outbound e–mail addresses. To create the new configuration, you need to understand the purpose of class M and macro M, both of which are found in the Local Info section of the sendmail.cf file.

sendmail calls hiding the real hostname *masquerading*. Thus, the name of the macro used to rewrite the sender host address is M. Set M to the domain name to replace the name of the local host in outbound mail with the name of the domain. Class M defines other hostnames, not just the local hostname, that also should be rewritten to the value of macro M. Class M is used on mail servers that need to rewrite sender addresses for their clients.

Checking the Red Hat sendmail.cf file on parrot, you find that no value is assigned to macro M, which means that masquerading is not being used. Further, you find that there is no class M declaration in the file. To masquerade the local host as foobirds.org and to masquerade the outbound mail from the clients robin and puffin, copy the sendmail.cf file to test.cf and then edit test.cf, changing the macro M declaration and adding a class M declaration:

```
# who I masquerade as (null for no masquerading)
DMfoobirds.org
# class M: host names that should be converted to $M
CMpuffin.foobirds.org robin.foobirds.org
```

Given these macro M and class M definitions, parrot rewrites its own outbound mail to *user*@foobirds.org, as well as rewriting mail from *user*@puffin.foobirds.org or user@robin.foobirds.org to user@foobirds.org. parrot is a mail server. Although you might use macro M on any system, you won't use class M on any type of system except a mail server.

A problem with using class M is that kathy@puffin.foobirds.org, kathy@robin.foobirds.org, and kathy@parrot.foobirds.org are all rewritten as kathy@foobirds.org. That's great if there really is only one kathy in the entire domain; otherwise, this may not be what you want. Coordinate usernames carefully across all systems. It simplifies the configuration of several different applications.

After setting a value for the M macro in the test.cf file, run a test to see if it works. Running sendmail with the test configuration does not affect the sendmail daemon that was started by the boot script. A separate instantiation of sendmail is used for the test.

Testing Your New Configuration

Test whether or not the change made to macro M in the configuration files modifies the rewrite process by directly testing the rewrite rulesets. First, you need to find out what rules are used to rewrite the address.

Use your knowledge of the flow of rulesets from Figure 5.1 to determine which rulesets to test. You know that the ruleset canonify is applied to all addresses. It is followed by different rulesets, depending on whether the address is a delivery address, a sender address, or a recipient address. Furthermore, the rulesets used for sender and recipient addresses vary, depending on the mailer that is used to deliver the mail. All addresses are then processed by ruleset final.

There are two variables that determine the rulesets used to process an address: the type of address and the mailer through which it is processed. The three address types are delivery address, recipient address, and sender address. You know the address type because you select the address being tested. In the example, the concern is the sender address.

There are two types of sender addresses: the sender address in the message header and the sender address in the "envelope." The message header address is the one on the From line sent with the mail. You probably see it in the mail headers when you view the message with your mail reader. The "envelope" address is the address used during the SMTP protocol interactions. The one that we're interested in is the one that remote users see in the mail—the header address.

Locating the Correct Mailer

The other variable that determines the rulesets used to process an address is the mailer. To find out which mailer delivers the mail, run sendmail with the –bv argument:

```
#$ sendmail -bv craig@wrotethebook.com
craig@wrotethebook.com... deliverable: mailer esmtp, host
wrotethebook.com., user craig@wrotethebook.com
```

To see which mailer is used to deliver mail to remote sites, run sendmail with the –bv argument, and give it a valid e–mail address for the remote site. In the example, the address is craig@wrotethebook.com. sendmail displays the mail delivery triple returned by ruleset parse: the mailer, the host, and the user. From this, you know that the mailer is esmtp.

Testing How Addresses Are Rewritten

To test the new configuration, run sendmail with the –bt option. sendmail displays a welcome message, and waits for you to enter a test. A simple test is a list of ruleset names followed by an e–mail address. For example, entering **canonify,parse craig** at the prompt would process the e–mail address craig through the rulesets canonify and then parse. This should provide you with the mailer, host, user delivery triple for the address.

Because you know the mailer that you want to test, you can use the /try command at the prompt to process the sender From address for the smtp mailer. The example in Listing 5.6 illustrates the test. First, test the existing configuration to see how the address is processed by the default configuration.

Listing 5.6: Testing the Default *sendmail* Configuration

```
# sendmail -bt
```

```
ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
> /tryflags HS
> /try smtp craig
Trying header sender address craig for mailer esmtp
canonify       input: craig
Canonify2      input: craig
Canonify2    returns: craig
canonify     returns: craig
1              input: craig
1            returns: craig
HdrFromSMTP    input: craig
PseudoToReal   input: craig
PseudoToReal returns: craig
MasqSMTP       input: craig
MasqSMTP     returns: craig < @ *LOCAL* >
MasqHdr        input: craig < @ *LOCAL* >
MasqHdr      returns: craig < @ parrot . foobirds . org . >
HdrFromSMTP  returns: craig < @ parrot . foobirds . org . >
final          input: craig < @ parrot . foobirds . org . >
final        returns: craig @ parrot . foobirds . org
Rcode = 0, addr = craig@parrot.foobirds.org
> /quit
```

Run sendmail –bt, which starts sendmail in test mode with the default configuration. Listing 5.6 shows exactly how the standard configuration processes e–mail addresses. Specifically, it shows how local sender addresses are rewritten for outbound mail.

The /tryflags command defines the type of address to be processed. Four flags are available: S for sender, R for recipient, H for header, and E for envelope. By combining two of these flags, the /tryflags command tells sendmail to process a header sender (HS) address.

The /try command tells sendmail to process the e–mail address craig through the mailer esmtp. The address returned by ruleset final, which is always the last ruleset to process an address, shows the address used on outbound mail after all of the rulesets have processed the address. With the default configuration, the input address craig is converted to craig@parrot.foobirds.org.

Next, run sendmail –bt with the –C option to use the newly created test.cf configuration file. The –C option permits you to specify the sendmail configuration file on the command line. Listing 5.7 shows this test.

Listing 5.7: Testing *sendmail* Masquerading

```
# sendmail -bt -Ctest.cf
ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
> /tryflags HS
> /try smtp craig
Trying header sender address craig for mailer esmtp
canonify           input: craig
Canonify2          input: craig
Canonify2        returns: craig
canonify         returns: craig
1                  input: craig
1                returns: craig
HdrFromSMTP        input: craig
PseudoToReal       input: craig
PseudoToReal     returns: craig
```

```
MasqSMTP           input: craig
MasqSMTP         returns: craig < @ *LOCAL* >
MasqHdr            input: craig < @ *LOCAL* >
MasqHdr          returns: craig < @ foobirds . org . >
HdrFromSMTP      returns: craig < @ foobirds . org . >
final              input: craig < @ foobirds . org . >
final            returns: craig @ foobirds . org
Rcode = 0, addr = craig@foobirds.org
> /quit
```

Running sendmail –bt –Ctest.cf starts sendmail in test mode, and tells it to use the new configuration that is stored in test.cf. The test in Listing 5.7 shows that the value entered in the M macro is used to rewrite the sender address in the message header. You know this because the address returned from ruleset final is now craig@foobirds.org, which is just what you want.

Run additional tests (for example, /try esmtp kathy@robin.foobirds.org) to see if client addresses are rewritten correctly. When you're confident that the configuration is correct and reliable, move the test.cf configuration file to sendmail.cf to make the new configuration available to sendmail.

If you are called upon to help someone configure sendmail on a system that doesn't already have the m4 source file installed, it may be easier to directly edit the sendmail.cf file, but only if the change is very small. If you can avoid it, don't make changes directly to the sendmail.cf file. If you really want to make major sendmail configuration changes, use m4 to build your configuration.

Using *m4* to Configure *sendmail*

The sendmail distribution contains m4 source files that build the sendmail.cf file. Sample m4 source files probably are included with your Linux system. If your Linux distribution doesn't include the m4 source files, you can download them from ftp://ftp.sendmail.org/, where they are stored as part of the latest sendmail distribution.

This section builds a custom sendmail.cf file using the m4 source files that come with a Red Hat system. On a Red Hat system, the m4 source files are in an RPM package separate from the package that includes the sendmail program. If your Red Hat system does not have the m4 source files, you need to install the RPM package. Figure 5.2 shows a gnorpm query for the sendmail–cf RPM file on our sample Red Hat system.

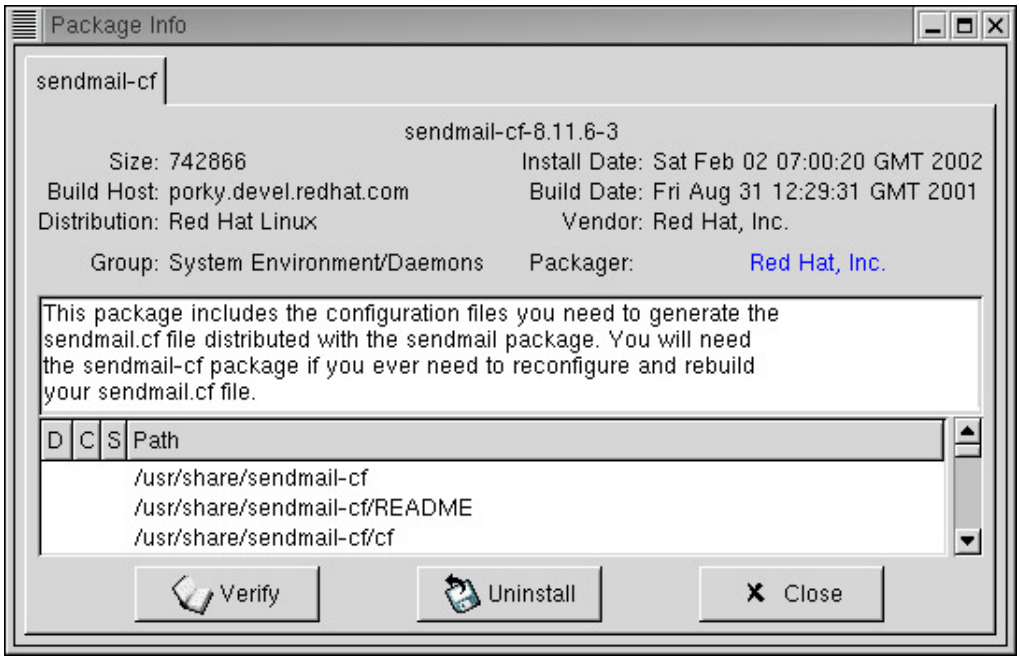


Figure 5.2: Contents of the *sendmail–cf* RPM

The sample configuration files are contained in the /usr/share/sendmail–cf/cf directory on our Red Hat system. Several of these are generic files preconfigured for different operating systems. The directory contains generic configurations for BSD, Solaris, SunOS, HP Unix, Ultrix, and (of course) Red Hat Linux. The Red Hat configuration is named redhat.mc. The directory also contains prototype files designed for use with any operating system. Despite the fact that there is a Red Hat source file, this book modifies the tcpproto.mc file. The tcpproto.mc file is a prototype configuration for any system on a TCP/IP network. It is not specific to Red Hat. It comes as part of the basic sendmail distribution, and can be modified for any operating system. It is a clean, vendor–neutral starting point for explaining m4 configuration. In reality, you will probably use the configuration provided by your vendor and will not build a configuration from scratch. However, it is good to know how to build a configuration from scratch if you ever need to, and the skills used to build a configuration are the same ones you will use to customize a vendor–provided configuration.

The *m4* Macro Control File

The /usr/share/sendmail–cf/cf directory's prototype files contain m4 macro commands. In addition to lots of comments, the tcpproto.mc file contains the macros shown in Listing 5.8.

Listing 5.8: The *tcpproto.mc* File

```
VERSIONID(`$Id: tcpproto.mc,v 8.13.22.1 2000/08/03 15:25:20 ca Exp $')
OSTYPE(`unknown')
FEATURE(`nouucp', `reject')
MAILER(`local')
MAILER(`smtp')
```

Listing 5.8 shows the configuration macros. The file tcpproto.mc also contains divert and dnl commands. A divert(–1) command precedes a large block of comments. m4 skips everything between a divert(–1) command and the next divert command. A divert(0) command ends the block of comments. Lines following a divert(0) command are processed by m4. The dnl command is used for single–line and partial–line comments. Everything after a dnl command up to the next newline character is skipped. If dnl appears at the beginning of a line, the entire line is treated as a

comment. The comments from the tcpproto.mc file are not included in Listing 5.8.

The VERSIONID macro defines version–control information. The version–control information can be anything you wish. Normally, it is information significant to some version control package. This macro is optional, and is just ignored in this discussion.

The OSTYPE macro loads the m4 source file from the ../ostype directory that defines the operating system information. The ostype directory contains more than 40 predefined operating system macro files, one of which is linux.m4. The OSTYPE macro is required.

The FEATURE macro defines an optional sendmail feature. The nouucp feature in the tcpproto.mc prototype file means that no UUCP address processing code will be included in the sendmail configuration. Later, we will add our own FEATURE macros to create a complete custom configuration.

The MAILER macros are the last macros in the sample file. The MAILER(local) macro adds the local mailer and the prog mailer to the configuration. The MAILER(smtp) macro adds mailers for SMTP, Extended SMTP, 8–bit SMTP, on–demand SMTP, and relayed mail. All of these mailers were described earlier in the chapter, and are all of the mailers needed for the configuration.

Creating a sendmail.cf for a Linux system from the tcpproto.mc prototype file could be as simple as changing the OSTYPE line from unknown to linux and then processing the file with the m4 command. The sendmail.cf file output by m4 would be ready for sendmail. In fact, it would be almost identical to the linux.smtp.cf configuration file delivered with Slackware Linux. Every custom configuration for Linux must begin by setting OSTYPE to linux in order to process the linux.m4 source file.

The Linux OSTYPE File

The OSTYPE file contains operating system–specific configuration values. The most common configuration variation between the different operating systems that run sendmail is the location of files. Variables that define pathnames are commonly stored in the OSTYPE file. However, any valid m4 macro can be placed in the OSTYPE file.

The command OSTYPE(linux) loads a file named linux.m4 from the ostype directory. On our sample Red Hat system, the full path of this directory is /usr/share/sendmail–cf/ostype. Listing 5.9 shows the contents of the linux.m4 file.

Listing 5.9: The *linux.m4* OSTYPE File

```
divert(0)
VERSIONID(`$Id: linux.m4,v 8.11.16.2 2000/09/17 17:04:22 gshapiro  Exp $')
define(`confEBINDIR', `/usr/sbin')
ifdef(`PROCMAIL_MAILER_PATH',,
        define(`PROCMAIL_MAILER_PATH', `/usr/bin/procmail'))
FEATURE(local_procmail)
```

The file begins with a block of comments that are bracketed by a divert(–1) statement and a divert(0) statement. The comments were deleted from Listing 5.9 to reduce the size of the listing. The VERSIONID macro can also be ignored.

The first real configuration command in the file is a define statement that assigns a value to the confEBINDIR parameter. This parameter stores the path of the directory that holds certain executable binary files. The sendmail default for confEBINDIR is /usr/libexec. This define changes the setting to /usr/sbin. Both of these directories exist on Linux systems, but the /usr/sbin directory is the one that is more commonly used to hold system binary files; and in this case, it is the correct setting. The confEBINDIR path is used to locate the smrsh program, which is frequently used as the prog mailer on Linux systems. A couple of quick ls commands on our sample Linux system show that the correct value for confEBINDIR is /usr/sbin:

```
$ ls /usr/libexec/smrsh
ls: /usr/libexec/smrsh: No such file or directory
$ ls /usr/sbin/smrsh
/usr/sbin/smrsh
```

The second configuration command is also a define. This one is a little more complex. This define is contained inside an ifdef. ifdef is a built–in m4 conditional command that checks whether or not a variable has already been set to a value. The ifdef command has three fields:

- the name of the variable that is being tested
- the action to take if the variable has been set
- the action to take if the variable has not been set

In Listing 5.9, the ifdef tests the variable PROCMAIL_MAILER_PATH. If the variable has already been defined, nothing is done. We know this by the fact that the second field of the ifdef is empty—notice the two commas right in a row (,,). If the variable has not yet been set, the define contained in the third field of the ifdef is executed.

The define assigns the variable PROCMAIL_MAILER_PATH the path value /usr/bin/procmail. This overrides the sendmail default for PROCMAIL_MAILER_PATH, which is /usr/local/bin/ procmail. Again, a quick ls shows that the new value is correct for our sample system:

```
$ ls /usr/bin/procmail
/usr/bin/procmail
```

As Listing 5.9 shows, the last line in linux.m4 is a FEATURE macro. The feature that this macro adds to the configuration is local_procmail, which causes procmail to be used as the local mailer. procmail is a very powerful mailer. The fact that Linux uses procmail as the local mailer is a plus.

The linux.m4 OSTYPE file defines the directory path for the smrsh program, the path for procmail, and a feature that uses procmail as the local mailer. The OSTYPE file is a good place to set file pathnames and mailer options that are specific to the operating system. Everything in the linux.m4 file is valid for all Linux systems.

In addition to creating a configuration that will run under Linux, we want to create a configuration that is customized for our organization. Assume that we want to create a custom configuration that converts *user@host* e–mail addresses into *firstname.lastname@domain* addresses. For that, we create a second m4 source file loaded from the domain directory. Let's look at that file in detail.

Creating an *m4* DOMAIN File

The domain directory is intended for m4 source files that contain information specific to your domain. This is a perfect place to put the commands that rewrite the hostname to the domain name on outbound mail, so we create a new m4 macro file in the domain directory and call it foobirds.m4.

We begin by changing to the ../domain directory and copying the file generic.m4 to foobirds.m4 to act as a starting point for the configuration. Listing 5.10 shows these steps.

Listing 5.10: The *generic.m4 DOMAIN* File

```
# cd /usr/share/sendmail-cf/domain
# cp generic.m4 foobirds.m4
# chmod 644 foobirds.m4
# tail -6 foobirds.m4
VERSIONID(`$Id: generic.m4,v 8.15 1999/04/04 00:51:09 ca Exp $')
define(`confFORWARD_PATH', `$z/.forward.$w+$h:$z/.forward+$h
Ä:$z/.forward.$w:$z/.
forward')dnl
define(`confMAX_HEADERS_LENGTH', `32768')dnl
FEATURE(`redirect')dnl
FEATURE(`use_cw_file')dnl
EXPOSED_USER(`root')
```

After copying generic.m4 to foobirds.m4, use the chmod command to set the file access permissions for the new file. (On some systems, the files in the domain directory are read–only.) Make the file read and write for the owner, and read–only for the group and the world.

Note See Chapter 9, "File Sharing," for more information on Linux file permissions.

The tail command displays the last six lines in the newly created foobirds.m4 file. (All of the lines before this are comments that are of no interest for this discussion.) You have already seen the VERSIONID macro. The first new line is the macro that defines confFORWARD_PATH, which tells sendmail where to look for the user's .forward file. The $z and the $w sendmail variables were described earlier in the discussion of the sendmail.cf ForwardPath option, which contained the two paths $z/.forward.$w:$z/.forward. Those two paths are the default value for confFORWARD_PATH. The define in Listing 5.10 increases the complexity of the.forward path list by adding the value $z/.forward.$w+$h:$z/.forward+$h to the default search list. The $z and $w variables serve the same purpose as before. The $h variable contains the *detail* value when the *user+detail* addressing syntax is used, and procmail is used as the local mailer. We know that procmail is being used as the local mailer from the local_procmail feature in the linux.m4 OSTYPE file. Given this specific configuration, local mail on a host named egret addressed to craig+sybex would prepend the following .forward search path to the standard path: /home/craig/.forward.egret+sybex:/home/craig/.forward+sybex. Even though *user+detail* addressing is rarely used, we decide to keep this define in the configuration because we plan to use procmail as the local mailer.

The second define sets the maximum number of bytes allowed for all headers on any one piece of mail. By default, no limit is set. In Listing 5.10, the maximum length is set to 32,768 bytes (32KB), which is more than enough for any reasonable set of headers. Headers longer than that might indicate a mail problem or some form of mail abuse. So we will keep this setting.

The FEATURE(redirect) macro adds support for the .REDIRECT pseudo–domain. The .REDIRECT pseudo–domain handles mail for people who no longer read mail at your site, but who still get mail sent to an old address. After enabling this feature, add aliases for each obsolete mailing address in the form:

```
old-address        new-address.REDIRECT
```

For example, assume that Jay Henson is no longer a valid e–mail user in your domain. His old username, jay, should no longer accept mail. His new mailing address is HensonJ@industry.com. Enter the following alias in the /etc/aliases file:

```
jay                 HensonJ@industry.com.REDIRECT
```

Now when mail is addressed to the jay account, the following error is returned to the sender telling them to try a new address for the recipient:

```
551 User not local; please try <HensonJ@industry.com>
```

This seems like a useful feature, so we keep it in the configuration.

The next line in the file also defines a useful feature. FEATURE(use_cw_file) is equivalent to the Fw/etc/local–host–names command in the sendmail.cf file. As described earlier, the local–host–names file provides a means for defining host aliases, which allow a mail server to accept mail addressed to other hosts.

The last line in Listing 5.10 is the EXPOSED_USER macro. The EXPOSED_USER macro adds usernames to class E that are not to be masqueraded, even when masquerading is enabled. Some usernames, such as root, occur on many systems, and are therefore not unique across a domain. For those usernames, converting the host portion of the address makes it difficult to sort out where the message really came from, and makes replies impossible. For example, assume that mail from root@wren.foobirds.org and root@ibis.foobirds.org is passed through a server that converts both addresses to root@foobirds.org. There is no way for the recipient to know exactly where the message really originated, and the remote user could not reply to the correct address. The EXPOSED_USER command prevents that from happening by ensuring that root is not masqueraded.

We discussed hostname masquerading earlier in this chapter when we covered class M. But now we are building a new configuration, and so far we have done nothing to enable masquerading. However, we keep the EXPOSED_USER macro in the foobirds.m4 file because we plan to add masquerading as part of the custom address processing for our domain. To the configuration commands shown in Listing 5.10, we add the following lines to perform the special address processing that we want.

```
MASQUERADE_AS(foobirds.org)
FEATURE(masquerade_envelope)
FEATURE(genericstable)
```

The MASQUERADE_AS line tells sendmail to hide the real hostname, and display the name foobirds.org in its place in outbound e–mail addresses. This defines the sendmail M macro that was used earlier in the chapter. The M macro only masqueraded header sender addresses. To do this on "envelope" addresses as well as message header addresses, use the FEATURE(masquerade_envelope) macro. The other FEATURE macro tells sendmail to use the generic address conversion database to convert login usernames to the value found in the database. This allows much more freedom in rewriting outbound addresses than was possible by directly modifying the sendmail.cf file. Listing 5.11 shows the completed foobirds.m4 DOMAIN file. Note that we also updated the data on the VERSIONID command line.

Listing 5.11: A Customized DOMAIN File

```
# cat foobirds.m4
divert(0)
```

```
VERSIONID(`foobirds.m4 03/16/2002')
define(`confFORWARD_PATH', `$z/.forward.$w+$h:$z/.forward+$h:$z/
.forward.$w:$z/.
forward')dnl
define(`confMAX_HEADERS_LENGTH', `32768')dnl
FEATURE(`redirect')dnl
FEATURE(`use_cw_file')dnl
EXPOSED_USER(`root')
MASQUERADE_AS(foobirds.org)
FEATURE(masquerade_envelope)
FEATURE(genericstable)
```

Building the *m4* Configuration File

Now that the m4 source files have been created for the operating system and the domain, create a new m4 configuration file to use them. All of the m4 macros related to rewriting the outbound addresses are in the foobirds.m4 file. The macros that are specific to the Linux distribution are in the linux.m4 file. We need to include those files in the configuration.

Begin by changing to the ../cf directory and copying the tcpproto.mc file to linux.mc. Then change the file permission for linux.mc to 644 to make sure that the file is writable by the owner.

Now, modify the file to reflect the new configuration. To do that, change "unknown" in the OSTYPE macro to "**linux**", and add a DOMAIN(foobirds) line to the linux.mc macro control file. For the sake of clarity, we also change the information on the VERSIONID line. The tail command in Listing 5.12 shows the macros in the edited file.

Listing 5.12: A Customized Macro Control File

```
# tail -7 linux.mc
divert(0)dnl
VERSIONID(`linux.mc 03/16/2002')
OSTYPE(`linux')
DOMAIN(`foobirds')
FEATURE(`nouucp', `reject')
MAILER(`local')
MAILER(`smtp')
```

The next step is to process the linux.mc file through m4:

```
# m4 ../m4/cf.m4 linux.mc > linux.cf
```

The sample shows the m4 command format used to build a sendmail.cf file. The pathname ../m4/cf.m4 is the path to the m4 source tree required to build a sendmail.cf file. The new macro control file is, of course, linux.mc. m4 reads the source files ../m4/cf.m4 and linux.mc, and it outputs the file linux.cf. The file output by the m4 command is in the correct format for a sendmail.cf file.

We used three files—ostype/linux.m4, domain/foobirds.m4, and cf/linux.mc—which together total less than 30 lines. These files create a sendmail.cf file that contains more than 1000 lines. The m4 macros are clearly the best way to build a custom sendmail configuration.

Building a *sendmail* Database

The configuration we have just created works fine. It operates just like the sendmail.cf that was created earlier, including masquerading hostnames as foobirds.org. But we also want to convert the username part of outbound addresses from the login name to the user's real name written as *firstname.lastname*. To do that, create a database to convert the username part of outbound e–mail addresses. Build the database by creating a text file with the necessary data and processing that file through the makemap command that comes with the sendmail distribution.

The genericstable database is a database that sendmail uses to convert outbound e–mail addresses. By default, the file is built in the /etc/mail directory. Listing 5.13 shows a simple genericstable file.

Listing 5.13: A Sample *genericstable*

```
pat Pat.Stover
mandy Amanda.Jenkins
kathy Kathy.McCafferty
sara Sara.Henson
norm Norman.Edwards
craig Craig.Hunt
```

Every entry in the genericstable database has two fields: The first field is the key, and the second is the value returned by the key. In the sample database, the key is the login name, and the return value is the user's real name. Using this database, a query for "pat" will return the value "Pat.Stover."

Before the genericstable can be used by sendmail, it must be converted from a text file to a database with the makemap command, which is included in the sendmail distribution. Assume that the data shown in Listing 5.13 are stored in a file named usernames.txt. The following command would convert that file to a genericstable database:

```
# makemap hash genericstable < usernames.txt
```

The sample makemap command creates a hash type database, which is the most commonly used. makemap can create other types of databases, but hash is the default type that sendmail uses for most databases.

After this genericstable database is created, login names on outbound mail are converted to full names. For example, the username mandy is converted to Amanda.Jenkins. Combining this with domain name masquerading rewrites outbound addresses into the *firstname.lastname@domain* format.

Testing the *m4* Configuration

Test the new configuration using the sendmail –bt command exactly as it was used earlier in this chapter. Listing 5.14 shows a test of the linux.cf file that we built with m4 macros.

Listing 5.14: Testing Address Rewriting

```
# sendmail -bt -Clinux.cf
ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
```

```
> /tryflags HS
> /try smtp craig
Trying header sender address craig for mailer esmtp
canonify         input: craig
Canonify2        input: craig
Canonify2      returns: craig
canonify       returns: craig
1                input: craig
1              returns: craig
HdrFromSMTP      input: craig
PseudoToReal     input: craig
PseudoToReal   returns: craig
MasqSMTP         input: craig
MasqSMTP       returns: craig < @ *LOCAL* >
MasqHdr          input: craig < @ *LOCAL* >
canonify         input: Craig . Hunt @ *LOCAL*
Canonify2        input: Craig . Hunt < @ *LOCAL* >
*LOCAL*: Name server timeout
Canonify2      returns: Craig . Hunt < @ *LOCAL* >
canonify       returns: Craig . Hunt < @ *LOCAL* >
MasqHdr        returns: Craig . Hunt < @ foobirds . org . >
HdrFromSMTP    returns: Craig . Hunt < @ foobirds . org . >
final            input: Craig . Hunt < @ foobirds . org . >
final            returns: Craig . Hunt @ foobirds . org
Rcode = 75, addr = Craig.Hunt@foobirds.org
> /quit
```

This time, when the sender address craig is processed through the Extended SMTP mailer, the address is rewritten to craig.hunt@foobirds.org using the genericstable database that we created. Again, after running several tests, copy linux.cf to /etc/sendmail.cf.

Of course, this entire configuration depends on having the m4 source files on the Linux system. If your system doesn't have the m4 source files, you can download the latest sendmail distribution from ftp://ftp.sendmail.org/, where it is stored in the pub/sendmail directory. The distribution includes a complete sendmail m4 source tree.

In Sum

sendmail is just the first step in building a fully functional mail server. Chapter 11 returns to this topic, and looks at several other software systems that are used to provide service to e−mail clients.

The next chapter, "The Apache Web Server," takes an in−depth look at a web server configuration. Web service is as important to a corporate network service as e−mail is as a user service.

Chapter 6: The Apache Web Server

Overview

For most people, the World Wide Web has become synonymous with the Internet. No discussion of Internet services is complete without mention of web servers. Web servers have become an essential part of every networked business—they are used to advertise products and offer services to external customers as well as to coordinate and disseminate information within the organization.

Linux systems make excellent web servers. In fact, the Apache server software that comes with Linux is the most widely used web server in the world. The daemon that Apache installs on a Linux system to create a web server is the Hypertext Transport Protocol daemon (httpd). This chapter describes how you can create your own web server with Apache and Linux by installing Apache and running httpd. It provides all the information you need to understand a default Apache configuration and to make the adjustments needed for an average server. For more advanced needs, see *Linux Apache Web Server Administration*, by Charles Aulds (Sybex, 2000).

Installing Apache

The Apache web server is part of most Linux distributions, and that includes the Red Hat Linux distribution that we are using as an example. The Apache web server software is one of the components that can be selected during the operating system installation. See Appendix A, "Installing Linux," for a description of this procedure.

If Apache is not among the software you selected during the initial installation, you need to install it now. The easiest way to install software is with a package manager. There are a couple that are available, but the most popular (and the one used on our sample Red Hat system) is the RPM Package Manager (RPM). In earlier chapters, we saw an example of the X–based Gnome RPM tool. RPM can also be used from the command line to manage the installation of optional software.

Use the rpm command to install the software you need, remove software you don't want, and check what software is installed in your system. rpm has many possible options, but most of them are for the developers who build the packages you want to install. For a network administrator, rpm can be reduced to three basic commands:

- rpm –i *package*: The –i option is used to install software.
- rpm –e *package*: The –e option is used to remove software.
- rpm –q: The –q option is used to list a software package already installed in the computer. Use –qa to list all installed packages.

To find the Apache package delivered with the distribution, mount the Linux distribution CD–ROM, and look in the RPMS directory. Here is an example from our Red Hat system:

```
$ cd /mnt/cdrom/RedHat/RPMS
$ ls *apache*
apache-1.3.20-16.i386.rpm
apacheconf-0.8.1-1.noarch.rpm
```

This example assumes that the CD–ROM was mounted on /mnt/cdrom. It shows that two software packages related to Apache are included in the Red Hat distribution. One is the web server

software, and the other is an X Window System Apache configuration tool. Install apache–1.3.20–16.i386.rpm with this command:

```
# rpm -i  apache-1.3.20-16.i386.rpm
```

After installing the package, check that it is installed with another rpm command:

```
$ rpm -q apache
apache-1.3.20-16
```

This example shows Apache being installed from the Red Hat distribution CD–ROM. If your Linux distribution does not include the Apache software, or if you want the latest release, download Apache from the Internet. It is available on the network in RPM format and in binary format for systems that do not have rpm tools.

Apache software is available at httpd.apache.org in both source and binary forms. Open your browser to the Apache web page and select the Download link. Then select the Binaries link and the Linux link. This displays a list of tar files containing pre–compiled Apache software (see Figure 6.1).

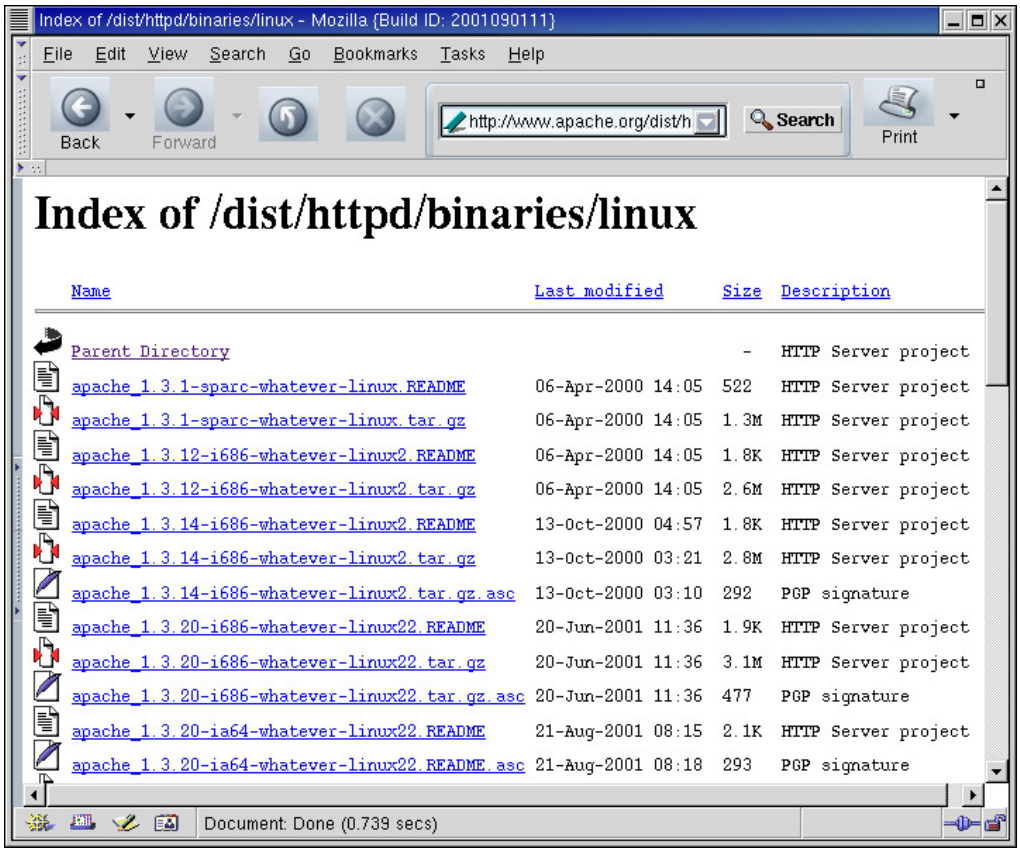


Figure 6.1: Linux binaries at the Apache website

The binaries are listed by "machine type." Linux runs on several different platforms. Select the binary that is appropriate for your processor. Use the Linux uname command to find out your server's machine type. For example, our sample system provides the following response:

```
$ uname -m
i686
```

Download the correct binary file to a working directory. Rename the current daemon so that it is not

accidentally executed in place of the new daemon, and move the new daemon into a directory reserved for third–party software. For example, on a Red Hat system, you might move the daemon to /usr/local/bin/httpd. Programs that are not managed by RPM should be installed in / usr/local or /opt. Otherwise, you impact some of the benefits of having a package manager. RPM can't verify or upgrade binaries that it doesn't manage. Placing a binary where RPM expects to find the binary that it manages can cause false RPM error messages, and may limit your ability to install RPM packages properly in the future. Here, we rename the old binary, copy the new one, and make sure to set the correct ownership and permissions for the file:

```
# mv /usr/sbin/httpd /usr/sbin/httpd.orig
# cp httpd /usr/local/bin/httpd
# chown root:root /usr/local/bin/httpd
# chmod 0755 /usr/local/bin/httpd
```

Running *httpd*

After the Apache RPM is installed, use a tool such as chkconfig or tksysv to add httpd to the boot process to ensure that the server restarts when the system reboots. For example, to start httpd for runlevels 3 and 5 on a Red Hat system, enter the following chkconfig command:

```
[root]# chkconfig --level 35 httpd on
[root]# chkconfig --list httpd
httpd           0:off   1:off   2:off   3:on    4:off   5:on    6:off
```

If your system doesn't have chkconfig, use another tool, such as tksysv. Figure 6.2 shows how tksysv is used to run httpd at startup. Highlight httpd in the Available box, click Add, and then click Done in the next two dialog boxes to add it to the startup process.

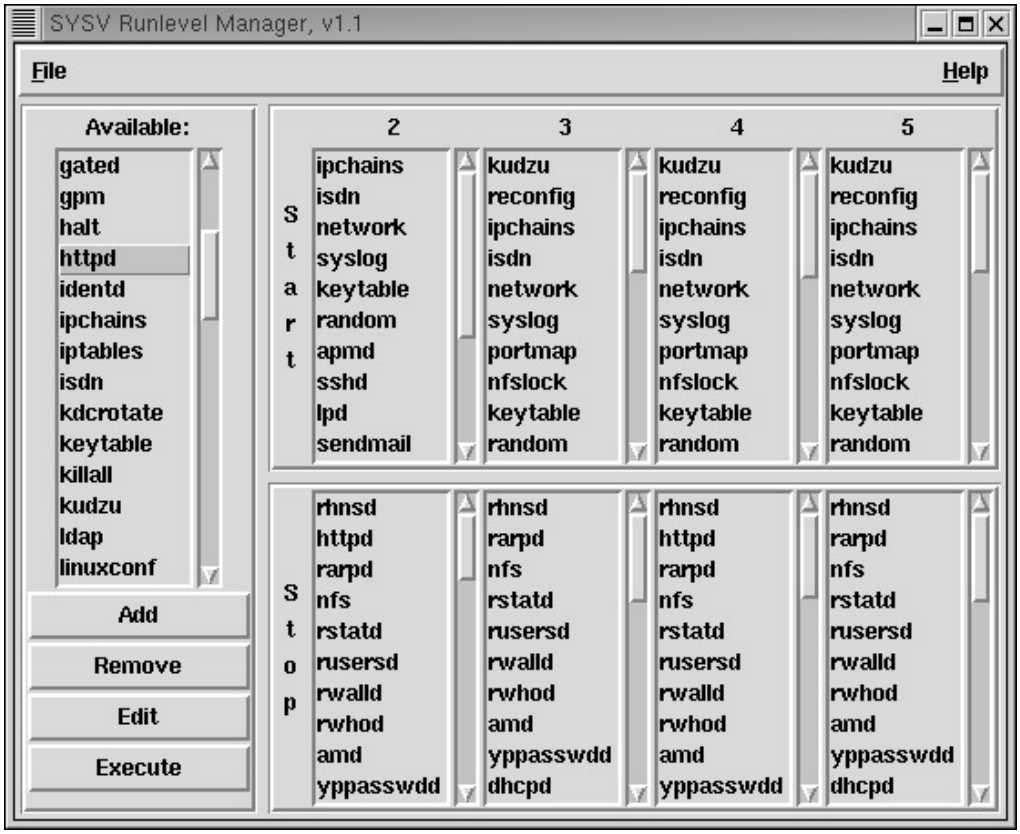


Figure 6.2: Enabling Apache with tksysv

Note

If you don't install Apache from an RPM file, you won't have the /etc/init.d/ httpd startup script, and you will need to add Apache to the startup on your own.

You might be surprised to find that Apache is already configured and ready to run. Try this little test.

Listing 6.1: Starting and Checking *httpd*

```
[root]# httpd &
[1] 2366
[root]# ps -Chttpd
  PID TTY          TIME CMD
 2367 ?        00:00:00 httpd
 2368 ?        00:00:00 httpd
 2369 ?        00:00:00 httpd
 2370 ?        00:00:00 httpd
 2371 ?        00:00:00 httpd
 2372 ?        00:00:00 httpd
 2373 ?        00:00:00 httpd
 2374 ?        00:00:00 httpd
 2375 ?        00:00:00 httpd
[1]+  Done                    httpd
```

Start the httpd daemon, and use the process status (ps) command to check for all httpd processes running on the system. (This group of httpd processes is called the swarm; we will cover the swarm in more detail later.)

Next, launch a web browser and point it to the localhost. Figure 6.3 shows the result. Not only is Apache installed and running, it is configured and responding with web data.

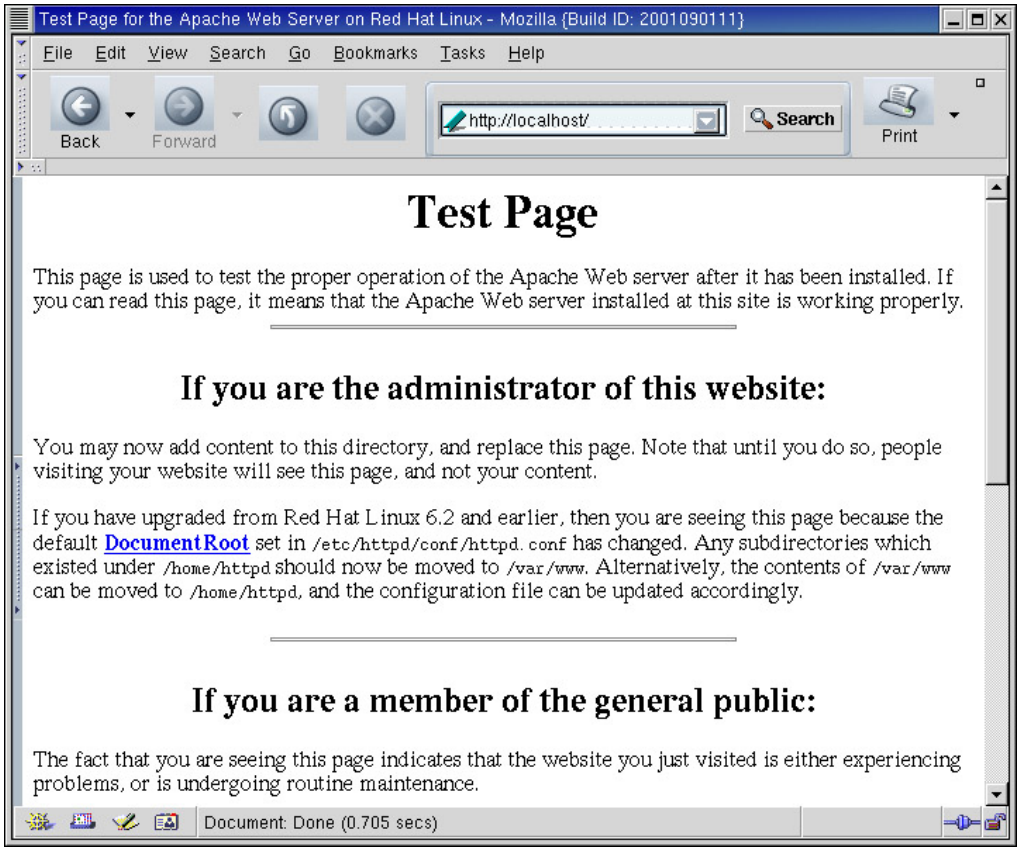


Figure 6.3: Apache installation web page

Configuring the Apache Server

Normally, at this point in the discussion of server software, I say something like this: "Installation is only the beginning. Now you must configure the software." That is not really the case for the Apache web server running on Linux. It is configured and will run with only a little input from you. You edit the httpd.conf file to set the web administrator's e−mail address in ServerAdmin and the server's hostname in ServerName. Beyond that, the httpd configuration provided with a Linux distribution should be adequate for that version of Linux.

The httpd.conf file is stored in the /etc/httpd/conf directory on Red Hat systems. Another operating system may place the configuration file in a different directory. To find where it is located on your system, look in the script that was used to start httpd. The location of the httpd.conf file is defined there. The locations of other files used by httpd are defined in httpd.conf. Another very simple way to locate the file is with the find command:

```
# find / -name httpd.conf -print
```

This command tells find to search every directory from the root (/) on down for a file named httpd.conf, and to print out the result of the search. Use find any time you need to locate a file.

After locating httpd.conf, use an editor to put valid ServerAdmin and ServerName values into the configuration. In the Red Hat Linux 7.2 example, ServerAdmin is delivered with this default value:

ServerAdmin root@localhost

The e−mail address of root@localhost is a valid address, but it is not one we would want to advertise to remote users. We change ServerAdmin to

ServerAdmin webmaster@www.foobirds.org

The delivered value for ServerName is:

#ServerName localhost

In this case, ServerName is commented out. We remove the hash mark (#) to activate the line, and we change ServerName to

ServerName www.foobirds.org

In Figure 6.3, we saw that when our Apache server is running, it is serving out data. Of course, the data in Figure 6.3 is not really the data we want to serve our clients. There are two solutions to this problem: either put the correct data in the directory that the server is using, or configure the server to use the directory in which the correct data is located.

The DocumentRoot directive points the server to the directory that contains web page information. By default, the Red Hat server gets web pages from the /var/www/html directory, as you can see by checking the value for DocumentRoot in the httpd.conf file:

```
$ grep '^DocumentRoot' httpd.conf
DocumentRoot "/var/www/html"
$ ls /var/www/html
index.html  manual  poweredby.png
```

The /var/www/html directory contains two files and one directory. The poweredby.png file is the "Powered by Red Hat Linux" graphic seen at the bottom of the web page that is shown in Figure 6.3. The index.html file is the HTML document that creates the web page seen in Figure 6.3. By default, Apache looks for a file named index.html, and uses it as the "home page" if a specific page has not been requested. The manual directory contains Apache documentation. It can be viewed by following the documentation link that is near the bottom of the default web page shown in Figure 6.3.

You can put your own index.html file in this directory, along with any other supporting files and directories you need, and Apache will start serving out your data. Alternatively, you can edit the httpd.conf file to change the value in the DocumentRoot directive to point to the directory in which you store your data. The choice is yours. Either way, you need to create HyperText Markup Language (HTML) documents for the web server to display.

HTML and document design are beyond the scope of this book. Creating the content for a professional website is not a job that is usually assigned to the system administrator. The people who build professional websites are creative people with a good sense of design and an artistic eye. That's not most of the technical people I know. If it's not you, either, the best approach for creating a website for your business is to hire someone who has the proper talent to do it, in the same way that your company hires professionals to produce its advertising campaigns.

Sometimes, however, you may be called upon to create a website for an organization that cannot afford to hire a professional web designer. In that case, you should refer to *Linux Apache Web Server Administration*, by Charles Aulds (Sybex, 2000) for more information about HTML authoring tools.

After the minimal changes are made to the httpd.conf file, the server can be restarted. The easiest way to do this on a Red Hat system is to run the /etc/init.d/httpd script file with the argument restart. The /etc/init.d/httpd script accepts the same arguments as those described in Chapter 4, "Linux Name Servers," for the /etc/init.d/named script.

The *httpd.conf* File

Despite the fact that you may change only one or two options, you need to understand how Apache is configured, what files the server requires, and where all the files are located. Given the importance of web servers for most networks, Apache is too essential for you to ignore. Additionally, the assumptions made by the distribution may not match the use you plan for your server, and you may want to fine–tune the configuration. To master Apache, you need to understand the Apache configuration file. Traditionally, Apache was configured by three files:

httpd.conf Defines configuration settings for the HTTP protocol and for the operation of the server. This includes defining what directory holds the configuration files.

srm.conf Configures how server requests are managed. This includes defining where HTTP documents and scripts are stored.

access.conf Defines access control for the server and the information it provides.

The functions of the three files overlap, and any Apache configuration directive can be used in any of the configuration files. The traditional division of the files into server, data, and security functions

was essentially arbitrary. Some administrators still follow this tradition, but it is most common for the entire configuration to be defined in the httpd.conf file. Placing the entire configuration in httpd.conf is the recommended approach, it is the approach used on our sample Red Hat system, and it is the one we use in this chapter. In the following sections, we examine the httpd.conf file in detail.

The httpd.conf file is an ASCII text file. Comments begin with a #, and the file is well–documented by comments. Most of the commands in the files are written in the form of a directive, followed by the value being assigned by the directive. For example:

Listen 443

This directive sets Listen to 443. (More about Listen later.)

In addition to basic directives, the httpd.conf file includes containers that limit the scope of the directives they contain. For example, to limit certain directives to a specific directory, create a Directory container for those directives. Five different types of containers found in the Red Hat httpd.conf are:

> **<Directory *pathname*>** The Directory directive creates a container for directives that apply to the directory identified by *pathname*. Any configuration directives that occur after the Directory directive and before the next </Directory> statement apply only to the specified directory.
>
> **<Files *filename*>** The Files directive creates a container for directives that apply to the file identified by *filename*. Any configuration directives that occur after the Files directive and before the next </Files> statement apply only to the specified file. *filename* can refer to more than one file because it can contain the wildcard characters * and ?. Additionally, if the Files directive is followed by an optional ~ (tilde), the *filename* field is interpreted as a regular expression.
>
> **<Location *document*>** The Location directive creates a container for directives that apply to a specific document. Any configuration directives that occur after the Location directive and before the next </Location> statement apply only to the specified document.
>
> **<IfDefine *argument*>** The IfDefine directive creates a container for directives that are applied to the configuration only if the specified argument exists on the httpd command line. For example, the line <IfDefine HAVE_SSL> marks the beginning of a container of directives that are used only if the string –DHAVE_SSL occurs on the httpd command line. The IfDefine container ends with the next </IfDefine> statement.
>
> **<IfModule *module*>** The IfModule directive creates a container for directives that are applied to the configuration only if the specified module is loaded. For example, the directives enclosed by <IfModule mod_userdir.c> and <\IfModule> are used only if the module mod_userdir is loaded.

Directories and files are easy to understand: They are parts of the filesystem that every system administrator knows. Documents, on the other hand, are specific to the web server. The screenful of information that appears in response to a web query is a document. It can be made up of many files from different directories. The Location container provides an easy way to refer to a complex document as a single entity. (We will see examples of Directory and Files containers later in this chapter.)

The Red Hat configuration contains many IfDefine and IfModule statements. The IfModule statements enclose commands that depend on the specific module; they allow the configuration to load without a syntax error, even if a specific module is not loaded. The IfDefine statements allow optional Apache features to be selected from the command line. The /etc/init.d/httpd script that starts httpd on a Red Hat system creates an argument for every module found in /usr/lib/ apache. Therefore, the default behavior on a Red Hat system is to attempt to use every optional feature included in /usr/lib/apache.

The Red Hat Linux httpd.conf file is more than 1400 lines long. But most of the contents of the file is information from the Apache developers that is designed to help you understand how to configure Apache. The file is full of comments that explain the purpose of the configuration directives, and additional information about the directives is available in the online documentation at http://www.apache.org/. Still, the Red Hat httpd.conf file contains more than 250 active configuration lines. To tackle that much information, we have organized the discussion of the configuration file into related topics. This is not the way the directives are organized inside the configuration file. The configuration file organizes directives by scope: global environment directives, main server directives, and virtual host directives. (Virtual hosts are explained later.) That organization is great for httpd when it is processing the file, but not so great for a human reading the file. We bring directives together into related groups to make the individual directives more understandable because after you understand the individual directives, you will understand the entire configuration.

We start our look at the httpd.conf file with the directives that load dynamically loadable modules. These modules must be loaded before the directives they provide can be used in the configuration, so it makes sense to discuss loading the modules before we discuss using the features they provide. Understanding dynamic loadable modules is a good place to start understanding Apache configuration.

Loading Dynamic Shared Objects

The two directives that appear most in the Red Hat httpd.conf file are LoadModule and AddModule. These two directives make up more than 75 of the 250 active lines in the httpd configuration file. All 75 of these lines configure the Dynamic Shared Object (DSO) modules used by the Apache web server.

Apache is composed of many software modules. Like kernel modules, DSO modules can be compiled into Apache or loaded at runtime. Run httpd with the –l command–line option to lists all of the modules compiled into Apache. Listing 6.2 shows the statically linked httpd modules on our sample Red Hat system.

Listing 6.2: Listing Statically Linked *httpd* Modules

```
$ httpd -l
Compiled-in modules:
  http_core.c
  mod_so.c
```

Some systems may have many modules compiled into the Apache daemon. Others, such as the Red Hat Linux 7.2 system, are delivered with only two modules compiled in. These are:

> **http_core.c** This is the core module. It is always statically linked into the Apache kernel. It provides the basic functions that must be found in every Apache web server. This module is required. All other modules are optional.

mod_so.c This module provides runtime support for Dynamic Shared Object modules. It is required if you plan to dynamically link in other modules at runtime. If modules are loaded through the httpd.conf file, this module must be installed in Apache to support those modules. For this reason, it is often statically linked into the Apache kernel.

Red Hat also uses many dynamically loaded modules. Two directives are used in the httpd.conf file to enable dynamically loaded modules. First, each module is identified by a LoadModule directive. For example, to request the module that handles the user agent log file, enter this line in the httpd.conf file:

```
LoadModule agent_log_module   modules/mod_log_agent.so
```

The LoadModule directive is followed by the module name and the path to the module itself.

In addition to the LoadModule directive, the Red Hat configuration identifies each module with an AddModule directive. This adds the module name to the list of modules that are actually loaded at runtime. The module list includes all optional modules—those that are compiled into the server and those that are dynamically loaded—except for the core module, which is not optional. For example, to add the agent_log_module to the module list, add the following line to the httpd.conf file:

```
AddModule mod_log_agent.c
```

The AddModule directive is followed by the source filename of the module being loaded. It is not the module name seen on the LoadModule line; it is the name of the source file that produced the object module, which is identical to the object filename except for the extension. On the LoadModule line, the object filename is mod_log_agent.so. Here, the source filename is mod_log_agent.c. Executable modules, called *shared objects*, use the extension .so, and the C–language modules in the add list use the extension .c.

Table 6.1 lists the modules that Red Hat Linux 7.2 identifies in its sample httpd.conf file with AddModule directives.

Table 6.1: DSO Modules Loaded in the Red Hat Configuration

Module	Purpose
mod_access	Specifies host– and domain–based access controls.
mod_actions	Maps a CGI script to a MIME file type.
mod_alias	Points to document directories outside the document tree.
mod_asis	Defines file types returned without headers.
mod_auth	Enables user authentication.
mod_auth_anon	Enables anonymous logins.
mod_auth_db	Enables use of a DB authentication file.
mod_autoindex	Enables automatic index generation.
mod_bandwidth	Sets bandwidth limits on server usage.
mod_cgi	Enables execution of CGI programs.
mod_dav	Provides WebDAV protocol extensions.
mod_dir	Controls formatting of directory listings.
mod_env	Allows CGI and SSI to inherit all shell environment variables.

mod_expires	Sets the date for the Expires header.
mod_headers	Enables customized response headers.
mod_imap	Processes image map files.
mod_include	Processes SSI files.
mod_info	Enables use of the server−info handler.
mod_log_agent	Points to the agent log file.
mod_log_config	Enables use of custom log formats.
mod_log_referer	Points to the referer log, which logs information about remote sites that refer to your site.
Module	Purpose
mod_mime	Provides support for MIME files.
mod_negotiation	Enables MIME content negotiation.
mod_perl	Provides support for the Perl language.
mod_php	Provides support for the PHP language.
mod_php3	Additional PHP support.
mod_php4	Additional PHP support
mod_put	Provides support for client to server file transfers using the PUT and DELETE commands.
mod_python	Provides support for the Python language.
mod_rewrite	Enables URI−to−filename mapping.
mod_roaming	Provides Netscape Roaming Access support.
mod_setenvif	Sets environment variables from client information.
mod_so	Provides runtime support for shared objects (DSO).
mod_ssl	Provides support for Secure Sockets Layer.
mod_status	Provides web−based access to the server−info report.
mod_throttle	Limits the maximum usage of individual users.
mod_userdir	Defines where users can create public web pages.
mod_vhost_alias	Provides support for name−based virtual hosts.

In addition to the LoadModule and AddModule directives, the Red Hat httpd.conf file contains one other directive that relates to loading DSOs. Before the modules are added to the list of modules that are available to Apache, the old module list can be cleared with the ClearModuleList directive. ClearModuleList occurs in the Red Hat httpd.conf file after the last LoadModule directive and before the first AddModule directive.

Basic Server Directives

A few directives define basic information about the server itself. We modified two of these, ServerAdmin and ServerName, when creating the basic configuration.

ServerAdmin defines the e−mail address of the web server administrator. In the default Red Hat configuration, this is set to root@localhost on the assumption that there is always a root account, and there is always the hostname localhost. Change this to the full e−mail address of the real web administrator. For example:

```
ServerAdmin webmaster@www.foobirds.org
```

In this example, we use the classic web administrator e-mail address webmaster@www.foobirds.org as the value for ServerAdmin. For this to work, we need a webmaster entry in the sendmail aliases file, which we created in Chapter 5, "Configuring a Mail Server"; and a CNAME record in the DNS database for www.foobirds.org, which we created in Chapter 4. Internet services are often interrelated and dependent on proper configuration in related services.

ServerName defines the hostname returned to clients when they read data from this server. In the default Red Hat configuration, ServerName was commented out, and the example on the comment line set ServerName to localhost. Change this to provide a real hostname. For example:

```
ServerName www.foobirds.org
```

When the ServerName directive is commented out, the "real" hostname is sent to clients. Thus, if the name assigned to the first network interface is wren.foobirds.org, it is the name sent to clients when ServerName is undefined. Defining an explicit value for ServerName documents the configuration, and ensures that you get exactly the value you want. We set ServerName to www.foobirds.org so that, even though the web server is running on wren, the server will be known as www.foobirds.org during web interactions. Of course, www.foobirds.org must be a valid hostname configured in DNS. In Chapter 4, we defined www is as a nickname for wren in the foobirds.org zone file.

The UseCanonicalName directive controls whether or not the value defined by ServerName is used. UseCanonicalName defines how httpd handles "self-referencing" URLs, which refer back to the server. When this is set to on, as it is in the Red Hat configuration, the value in ServerName is used. If it is set to off, the value that came in the query from the client is used. If your site uses multiple hostnames, you may want to set this to off so that the user will see the name they expect in the reply.

The ServerRoot option defines the directory that contains the httpd server files. This is different from DocumentRoot, which is the directory that contains the information files the server presents to clients. On Red Hat, and most other systems, this is /etc/httpd. The conf directory under the ServerRoot contains the three configuration files. Therefore, httpd.conf is itself located under the ServerRoot that it defines.

The ServerType option defines how the server is started. If the server starts from a startup script at boot time, the option is set to standalone. If the server is run on demand by inetd or xinetd, ServerType is set to inetd. Most of the time, web servers are in high demand, so it is best to start them at boot time. It is possible, however, for a user to set up a small, rarely used website on a Linux desktop. In that case, running the server from inetd or xinetd may be desirable.

Port defines the TCP port number used by the server. The standard number is 80. On occasion, private web servers run on other port numbers. 8080 and 8000 are popular alternative ports for private websites. If you change the number, you must then tell your users the non-standard port number. For example, http://private.foobirds.org:8080 is a URL for a website running on TCP port 8080 on host private.foobirds.org.

When ServerType is set to inetd, it is usually desirable to set Port to something other than 80. The reason for this is that the ports under 1024 are "privileged" ports. If 80 is used, httpd must be run from inetd with the user ID root. This is a potential security problem, because an intruder might be able to exploit the website to get root access. Using port 80 is okay when ServerType is standalone because the initial httpd process does not provide direct client service. Instead, it starts several other HTTP daemons to provide client services that do not run with root privilege.

Multi–Homed Server Configuration

A web server that is connected to more than one physical network is called a multi–homed server. Such a server has more than one IP address. If it does, the system needs to know which address it should listen to for incoming server requests. There are two configuration options to handle this:

> **BindAddress** Tells httpd which address should be used for server interactions. The default value is *, which means that the server should respond to web service requests addressed to any of its valid IP addresses. If a specific address is used on the BindAddress command line, only requests addressed to that address are honored. BindAddress is not explicitly set in the Red Hat configuration.
>
> **Listen** Tells httpd which additional addresses and ports should be monitored for web service requests. Address and port pairs are separated by a colon. For example, to monitor port 8080 on IP address 172.16.64.52, enter 172.16.64.52:8080. If a port is entered with no address, the address of the server is used. If the Listen directive is not used, httpd monitors only the port defined by the Port directive. The Red Hat configuration only uses the Listen directive to provide SSL support. In that case, it sets the standard port to 80, and sets the SSL port to 443.

Defining Where Things Are Stored

The DocumentRoot directive, which was mentioned earlier, defines the directory that contains the web server documents. For security reasons, this is not the same directory that holds the configuration files. The ServerRoot directive defines the location of the server configuration files. On our sample Red Hat system, DocumentRoot and ServerRoot are

```
DocumentRoot "/var/www/html"
ServerRoot "/etc/httpd"
```

The PidFile and ScoreBoardFile directives both define the paths of files that relate to process status. The PidFile is the file in which httpd stores its process ID; the ScoreBoardFile is the file in which httpd writes process status information. If the ScoreBoardFile is not defined, Apache uses a shared–memory segment instead of a file, which improves performance.

The Alias directive and the ScriptAlias directive both map a URL path to a directory on the server. For example:

```
Alias /icons/ "/var/www/icons/"
ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"
```

The first line maps the URL path /icons/ to the directory /var/www/icons/. Thus, a request for www.foobirds.org/icons/ is mapped to www.foobirds.org/var/www/icons/.

The Red Hat configuration contains several Alias directives to handle several different mappings, and one global ScriptAlias directive. The ScriptAlias directive functions in exactly the same ways as the Alias directive, except that the directory it points to contains executable CGI programs. Therefore, httpd grants this directory execution privileges. ScriptAlias is particularly important because it allows you to maintain executable web scripts in a directory that is separate from the DocumentRoot. CGI scripts are the single biggest security threat to your server. Maintaining them separately allows you to provide tighter controls on who has access to the scripts.

The UserDir directive enables personal user web pages, and points to the directory that contains the user pages. UserDir usually points to public_html. With this default setting, users create a directory named public_html in their home directories to hold their personal web pages. When a request comes in for www.foobirds.org/~sara, it is mapped to www.foobirds.org/home/sara/public_html .

An alternative is to define a full pathname such as /home/userpages on the UserDir command line. Then the administrator creates the directory, and allows each user to store personal pages in subdirectories of this directory. A request for www.foobirds.org/~sara maps to www.foobirds.org/home/homepages/sara . The advantages of this approach are that it improves security by making it easier for you to monitor the content of user pages, and it makes it easier to control who can publish pages, rather than allowing all users to do so.

The DirectoryIndex option defines the name of the file that is retrieved if the client's request does not include a filename. Our sample Red Hat system has the following values for this option:

```
DirectoryIndex index.html index.htm index.shtml index.php index.php4
   index.php3 index.phtml index.cgi
```

Given the value defined for DocumentRoot and this value, if the server gets a request for http://www.foobirds.org/songbirds/, it first attempts to locate a file named /var/www/html/songbirds/index.html. Notice that the DocumentRoot is prepended to any request, and the DirectoryIndex is appended to any request that doesn't end in a filename. If the server finds a file with that name, it serves the client the file. If it does not find the file, it tries index.htm and then index.shtml, and so on down the line to index.cgi. If none of the files defined by DirectoryIndex is found, httpd sends the client a listing of the directory. Several directives control how that directory listing is formatted.

Creating a Fancy Index

If the FancyIndexing option is specified on the IndexOptions directive, httpd creates a directory list that includes graphics, links, and other fancy features. The following options define the graphics and features used in the fancy directory listing:

IndexIgnore Lists the files that should not be included in the directory listing. Files can be specified by name, by partial name, by extension, or by standard wildcard characters.

HeaderName Defines the name of a file that contains information to be displayed at the top of the directory listing.

ReadmeName Defines the name of a file that contains information to be displayed at the bottom of the directory listing.

AddIcon Defines the icon file used to represent a file based on the filename extension.

DefaultIcon Defines the icon file used to represent a file that has not been given an icon by any other option.

AddIconByEncoding Defines the icon file used to represent a file based on the MIME encoding type of the file.

AddIconByType Defines the icon file used to represent a file based on the file's MIME file type.

Defining File Types

MIME file types and file extensions play a major role in helping the server determine how a file should be handled. Specifying MIME options is also a major part of the httpd.conf file. The options involved are the following:

DefaultType Defines the MIME type that is used when the server cannot determine the type of a file. By default, this is set to text/html. Thus, when a file has no file extension, the server assumes that it is an HTML file.

AddEncoding Maps a MIME encoding type to a file extension. The Red Hat configuration contains two AddEncoding directives:

```
AddEncoding x-compress Z
AddEncoding x-gzip gz tgz
```

The first directive maps the file extension .Z to the MIME encoding type x–compress. The second line maps the file extensions .gz and .tgz to MIME encoding type x–gzip.

AddLanguage Maps a MIME language type to a file extension.

LanguagePriority Sets the language encoding in case the client does not specify a preference.

AddType Maps a MIME file type to a file extension.

AddHandler Maps a file handler to a file extension. A *file handler* is a program that knows how to process a file. Simple examples of this are cgi–script, which is the handler for CGI files; and server–parsed, which handles Server Side Includes (SSI). (Both SSI and CGI are covered more later.)

Managing Child Processes

In the original NCSA (National Center for Supercomputer Applications) web server design, the server would fork processes to handle individual requests. This placed a heavy load on the CPU when the server was busy, and impacted the responsiveness of the server. It was even possible for the entire system to be overwhelmed by HTTP daemon processes.

Apache uses another approach. A swarm of server processes starts at boot time. (The ps command in Listing 6.1 shows several httpd processes running on a Linux system.) All of the processes in the swarm share the workload. If all of the persistent httpd processes become busy, spare processes are started to share the work.

Five options in the httpd.conf configuration file control how the child server processes are managed. The options that control the management of these spare processes are as follows:

MinSpareServers Defines the minimum number of idle server processes that must be maintained. In the Red Hat configuration, this is set to 5, which is the default value used in the Apache distribution. With this setting, another process is created to

maintain the minimum number of idle process when the number of idle httpd processes drops below 5. Set MinSpareServers higher if the server is frequently slow to respond because of periods of high activity.

MaxSpareServers Defines the maximum number of idle server processes that may be maintained. In the Red Hat configuration, this is set to 20. During a burst of activity, several httpd processes may be created to handle client request. As activity declines, the processes become idle. With the Red Hat setting, processes will be killed if more than 20 httpd processes are sitting idle.

StartServers Defines the number of persistent httpd processes started at boot time. In the Red Hat configuration it is set to 8. The effect of this directive can be seen in the output of the ps command in Listing 6.1, which showed that nine httpd daemons were running. One of these is the parent process that manages the swarm, but does not serve client requests. The other eight are the child processes that actually handle client requests for data.

MaxClients Defines the maximum number of clients to be serviced. Requests beyond the maximum number are queued until a server is available. Red Hat sets this to 150, which is the most commonly used value. The default used when MaxClients is not defined is the value set by HARD_SERVER_LIMIT when Apache is compiled, which is usually 256. MaxClients prevents the server from consuming all system resources when it receives an overwhelming number of client requests. The Red Hat MaxClients setting should be increased only if the number of simultaneous clients using your server routinely exceeds 150, and your server is a powerful system with fast disks and a large amount of memory. It is often better to handle additional clients by adding additional servers than it is to pack more clients on one server.

MaxRequestsPerChild Defines the number of client requests a child process can handle before it must terminate. Red Hat sets this to a very low 1000. MaxRequestsPerChild is used when the operating system or libraries have memory leaks that cause problems for persistent processes. Apache recommends a setting of 10000 if your system has memory leak problems. Set MaxRequestsPerChild to 0, which means "unlimited"—a child process can keep handling client requests for as long as the system is up and running—unless you know for a fact that the library you used to compile Apache has a memory leak.

The User and Group directives define the UID and GID under which the swarm of httpd processes are run. When httpd starts at boot time, it runs as a root process, binds to port 80, and then starts a group of child processes that provide the actual web services. The UID and GID defined in the httpd.conf file are assigned to these child processes. The UID and GID should provide the least possible system privilege to the web server. On most Linux systems, this is the user nobody and the group nobody. An alternative to using nobody is to create a user ID and group ID just for httpd. Red Hat uses this approach, and creates a special user apache and a special group apache. The advantage of creating a special UID and GID for the web server is that you can use group permission for added protection, and you won't be completely dependent on the world permission granted to nobody. If you create your own user and group for Apache, set the file permissions for the new user account very carefully. (See Chapter 9, "File Sharing," for information on filesystem security.)

Performance Tuning Directives

The KeepAlive directive enables or disables the use of persistent connections. Without a persistent connection, the client must make a new connection to the server for every link the client wants to explore. Because HTTP runs over TCP, every connection requires a connection setup. This adds time to every file retrieval. With a persistent connection, the server waits to see if the client has additional requests before it closes the connection. Therefore, the client does not need to create a new connection to request a new document. The KeepAliveTimeout defines the number of seconds the server holds a persistent connection open, waiting to see if the client has additional requests.

MaxKeepAliveRequests defines the maximum number of requests that will be accepted on a "kept–alive" connection before a new TCP connection is required. The Apache default value is 100. Setting MaxKeepAliveRequests to 0 allows unlimited requests. 100 is a good value for this parameter. Few users request 100 document transfers, so the value essentially creates a persistent connection for all reasonable cases. Additionally, if the client does request more than 100 document transfers, it might indicate a problem with the client system. At that point, requiring another connection request is probably a good idea.

Timeout defines the number of seconds the server waits for a transfer to complete. The value needs to be large enough to handle the size of the files your site sends and the low performance of the modem connections of your clients. But if it is set too high, the server will hold open connections for clients that may have gone offline. The Red Hat configuration has this set to five minutes (300 seconds).

BrowserMatch is a different type of tuning parameter: It reduces performance for compatibility's sake. The Red Hat configuration contains the following five BrowserMatch directives:

```
BrowserMatch "Mozilla/2" nokeepalive
BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0 force-response-1.0
BrowserMatch "RealPlayer 4\.0" force-response-1.0
BrowserMatch "Java/1\.0" force-response-1.0
BrowserMatch "JDK/1\.0" force-response-1.0
```

The BrowserMatch directives present information in a manner that is compatible with the capabilities of different web browsers. For example, a browser may be able to handle only HTTP 1.0, not HTTP 1.1. In this case, downgrade–1.0 is used on the BrowserMatch line to ensure that the server uses only HTTP 1.0 when dealing with that browser. In the Red Hat configuration, keepalives are disabled for two browsers. One browser is offered only HTTP 1.0 during the connection. And responses are formatted to be compatible with HTTP 1.0 for four different browsers.

Don't fiddle with the BrowserMatch directives. These settings are shipped as defaults in the Apache distribution. They are already set to handle the limitations of different browsers. These are tuning parameters, but they are used by the Apache developers to adjust to the limitation of older browsers.

Caching Directives

Several directives control the caching behavior of the server. A *cache* is a locally maintained copy of a server's web page. When firewalls are used, direct web access is often blocked. Users connect to a proxy server through the local network, and the proxy server is trusted to connect to the remote web server. Proxy servers do not have to maintain cached copies of web pages, but caching improves performance by reducing the amount of traffic sent over the WAN and by reducing the contention for popular websites. The directives that control caching behavior are as follows:

ProxyRequests Setting this option to on turns your server into a proxy server. By default, this is set to off.

ProxyVia Enables or disables the use of Via: headers, which aid in tracking where cached pages actually came from.

CacheRoot Defines the directory in which cached web pages are written. To avoid making the directory writable by the user nobody, create a special user ID for httpd when you run a proxy server.

CacheSize Defines the maximum size of the cache in kilobytes. The default is 5KB, which is a very minimal size. Many system administrators consider 100MB a more reasonable setting.

CacheGcInterval Defines the time interval at which the server prunes the cache. It is defined in hours, and the default is 4. Given the defaults, the server prunes the cache down to 5 kilobytes every four hours.

CacheMaxExpire Defines the maximum number of hours a document is held in the cache without requesting a fresh copy from the remote server. The default is 24 hours. With the default, a cached document can be up to a day out–of–date.

CacheLastModifiedFactor Defines the length of time a document is cached, based on when it was last modified. The default factor is 0.1. Therefore, if a document is retrieved that was modified 10 hours ago, it is held in the cache for only one hour before a fresh copy is requested. The assumption is that if a document changes frequently, the time of its last modification will be recent. Thus, documents that change frequently are cached only a short period of time. Regardless, nothing is cached longer than CacheMaxExpire.

CacheDefaultExpire Defines a default cache expiration value for protocols that do not provide the value. The default is one hour.

NoCache Defines a list of the hostnames of servers whose pages you do not want to cache. If you know that a server has constantly changing information, you don't want to cache information from that server because your cache will always be out–of–date. Listing the name of that server on the NoCache directive line means that queries are sent directly to the server, and responses from the server are not saved in the cache.

Defining Virtual Hosts

Virtual hosts allow a server to host multiple websites known by different hostnames. The Red Hat configuration has an entire section dedicated to virtual hosts, but it is all commented out. It is there only to serve as an example. To use virtual hosts, you must first uncomment the NameVirtualHost directive to enable name–based virtual hosts. There are IP–based virtual hosts, but those consume valuable IP addresses. Name–based virtual hosts are recommended by the Apache developers, and are preferred by most administrators.

On the sample Red Hat system, the NameVirtualHost directive is commented–out. The line is:

```
#NameVirtualHost *
```

The asterisk on this line stands for any address assigned to any interface on the host. To make this more understandable, we will be more explicit. We remove the hash mark (#) to activate the line, and set the NameVirtualHost address to the primary address of our server:

```
NameVirtualHost 172.16.5.1
```

Next, define the virtual hosts that will be served. For example, to host websites for fish.edu and mammals.com on the wren.foobirds.org server, add the following lines to the httpd.conf file:

```
<VirtualHost www.fish.edu>
DocumentRoot /var/www/html/fish
ServerName www.fish.edu
</VirtualHost>
<VirtualHost www.mammals.com>
DocumentRoot /var/www/html/mammals
ServerName www.mammals.com
</VirtualHost>
```

Each VirtualHost directive defines a hostname alias to which the server responds. For this to be valid, DNS must define the alias with a CNAME record. The example requires CNAME records that assign wren.foobirds.org the aliases of www.fish.edu and www.mammals.com. When wren receives a server request addressed to one of these aliases, it uses the configuration parameters defined here to override its normal settings. Therefore, when it gets a request for www.fish.edu, it uses www.fish.edu as its ServerName value instead of its own server name, and uses /var/www/ html/fish as the DocumentRoot.

These are simple examples. Any valid configuration directive can be placed within a VirtualHost container.

Web Server Security

Web servers are vulnerable to all of the normal security problems that are discussed in Chapter 12, "Security." But they also have their own special security considerations. In addition to all of the normal threats, such as network break–ins and denial of service attacks, web servers are responsible for protecting the integrity of the information disseminated by the server and for protecting the information sent by the client to the server.

Access to the server information is protected by access controls. Through the httpd.conf file, you can configure host–level and user–level access controls. Access control is important for protecting internal and private web pages, but most web information is intended for dissemination to the world at large. For these global web pages, you don't want to limit access in any way, but you do want to protect the integrity of the information on all pages.

One of the unique security risks for a web server is having an intruder change the information on the web pages. We have all heard of high–profile incidents when intruders get in and change the home page of some government agency, inserting comical or pornographic material. These attacks are not intended to do long–term harm to the server, but they are intended to embarrass the organization that runs the website.

Use the Linux file permissions discussed in Chapter 9 to protect the files and directories in which you store web documents. The server does not need write permissions, but it needs to read and execute these files. Executable files, if they are poorly designed, are always a potential security

threat.

The CGI and SSI Threat

Apache itself is very reliable and reasonably secure. The biggest threat to server security is the code that you or your users write for the server to execute. Two sources of these problems are Common Gateway Interface (CGI) programs and Server Side Includes (SSI).

One of the biggest threats to server security is badly written CGI programs. Intruders exploit poor code by forcing buffer overflows or by passing shell commands through the program to the system. The only way to avoid this and still have the benefit of CGI programs, which can be written in C, Perl, Python, and other programming languages, is to be very careful about the code that you make available on your system. Here are some basic preventative measures to keep in mind:

- Personally review all programs included in the cgi–bin directory.
- Try to write programs that do not allow free–form user input.
- Use drop–down menus instead of keyboard input.
- Limit what comes in to your system from the user.

To make it easier to review all CGI scripts, keep them all in the ScriptAlias directory. Don't allow ExecCGI in any other directory unless you're positive no one can place a script there that you have not personally reviewed. (The way ExecCGI and other server options are controlled is covered in the next section.)

Server Side Includes is also called Server Parsed HTML, and the files often have the .shtml file extension. These files are processed by the server before they are sent to the client. These files can include other files or execute code from script files. If user input is used to dynamically modify the SSI file, it is vulnerable to the same type of attacks as CGI scripts.

SSI commands are embedded inside HTML comments. Therefore, each SSI command begins with <!–– and concludes with ––>. The SSI commands are listed in Table 6.2.

Table 6.2: Server Side Includes Commands

Command	Purpose
#config	Formats the display of file size and time.
#echo	Displays variables.
#exec	Executes a CGI script or a shell command.
#flastmod	Displays the date a document was last modified.
#fsize	Displays the size of a document.
#include	Inserts another file into the current document.

The most secure way to operate a server is to disallow all SSI processing. This is the default unless All or Includes is specified by an Options directive in the httpd.conf file. A compromise setting is to allow SSI, but to disallow the #include and #exec commands, which are the greatest security threat. Use IncludesNOEXEC on the Options directive for this setting.

Server Options for Documents and Directories

The httpd.conf file can define server controls for all web documents or for documents in individual

directories. The Options directive specifies which server options are permitted for documents. Placing the Options directive inside a Directory container limits the scope of the directive to that specific directory. The Red Hat Linux 7.2 configuration provides the examples shown in Listing 6.3.

Listing 6.3: Active Directory Containers in Red Hat's *httpd.conf* File

```
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>
<Directory "/var/www/html">
    Options Indexes FollowSymLinks
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
<Directory "/var/www/icons">
    Options Indexes MultiViews
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
<Directory "/var/www/cgi-bin">
    AllowOverride None
    Options None
    Order allow,deny
    Allow from all
</Directory>
<Directory /usr/share/doc>
    order deny,allow
    deny from all
    allow from localhost .localdomain
    Options Indexes FollowSymLinks
</Directory>
```

This configuration defines server option controls for five directories: the root (/), /var/www/html, /var/www/icons, /var/www/cgi–bin, and /usr/share/doc directories. The example shows four possible values for the Options directive: FollowSymLinks, Indexes, None, and MultiView. The Options directive has several possible settings:

All Permits the use of all server options.

ExecCGI Permits the execution of CGI scripts from this directory. The ExecCGI option allows CGI scripts to be executed from directories other than the directory pointed to by the ScriptAlias directive. Many administrators set this option for the ScriptAlias directory, but doing so is somewhat redundant. The ScriptAlias directive already specifies that /var/ www/cgi–bin is the script directory. In Listing 6.3, Options is set to None for the /var/ www/cgi–bin directory without undoing the effect of the ScriptAlias directive.

FollowSymLinks Permits the use of symbolic links. If this is allowed, the server treats a symbolic link as if it were a document in the directory.

Includes Permits the use of Server Side Includes (SSI).

IncludesNOEXEC Permits Server Side Includes (SSI) that do not include #exec and #include commands.

Indexes Permits a server–generated listing of the directory if an index.html file is not found.

MultiViews Permits the document language to be negotiated.

None Doesn't permit any server options. This provides the highest level of security.

SymLinksIfOwnerMatch Permits the use of symbolic links if the target file of the link is owned by the same user ID as the link itself.

Use server options with care. The None, Indexes, and MultiView options used in the Red Hat configuration should not cause security problems, although Indexes gives remote users a listing of the directory contents if no index.html file is found and MultiView consumes server resources. FollowSymLinks has the potential for security problems because symbolic links can increase the number of directories in which documents are stored. The more directories, the more difficult the task of securing the directories because all of the directories must have the proper permissions set, and all must be monitored for possible file corruption.

The directory containers in the example above also contain AllowOverride directives. These directives limit the amount of configuration control given to the individual directories.

Directory–Level Configuration Controls

The httpd.conf directive AccessFileName .htaccess enables the use of a directory–level configuration control file, and states that the name of the directory configuration file is .htaccess. If the server finds a file with this name in a directory, it applies the configuration commands defined in the file to the directory based on the AllowOverride directive that applies to the directory. The .htaccess file allows you to distribute control to the individuals who create and manage the individual data directories.

Conceptually, the .htaccess file is similar to a Directory container. In the same way that the directives in a Directory container apply to a specific directory, the directives in the .htaccess file apply only to the directory in which the file is found. The directives in the .htaccess file are potentially the same ones used in the httpd.conf file that defines the system–wide configuration.

The AllowOverride directive has six keywords that set the level of configuration control granted to the .htaccess file:

None Allows no configuration overrides. In effect, None disables the .htaccess file.

All Permits the .htaccess file to override everything defined in the httpd.conf configuration files for which overrides are allowed. This is the same as specifying all of the four remaining keywords: AuthConfig, FileInfo, Indexes, and Limit.

AuthConfig Permits the .htaccess file to define user authentication directives. User authentication is covered later in this chapter.

FileInfo Allows the file to use directives that control document types.

Indexes Permits .htaccess to configure fancy indexing.

Limit Allows the file to configure host–level access controls. Access controls are discussed in the next section.

In addition to these keyword values, individual commands can be permitted through AllowOverride. For example, to allow a directory to define its own file extension mapping, specify

```
AllowOverride AddType
```

The Options and AllowOverride directives control access to server features and configuration overrides, which can help keep information safe from corruption. Sometimes, however, you have information you want to keep safe from widespread distribution. Access controls limit the distribution of information.

Defining Access Controls

In addition to the Options and AllowOverride directives, the Directory containers in Listing 6.3 enclose Order, Allow, and Deny directives. These three directives permit you to define host–level access controls. One example taken from Listing 6.3 will make this capability clear. Listing 6.4 shows a Directory container and an associated Alias directive.

Listing 6.4: Apache Access Controls

```
Alias /doc/ /usr/share/doc/
<Directory /usr/share/doc>
    order deny,allow
    deny from all
    allow from localhost .localdomain
    Options Indexes FollowSymLinks
</Directory>
```

This example shows an Alias directive that maps the document name "doc" to the directory /usr/share/doc, and the Directory container for /usr/share/doc, which is a directory on Linux systems that contains a wide range of documentation. The access controls in the Directory container allow users who are logged on to the web server to access the directory through a browser using the document name "doc". Users who are not directly logged into the server are denied access. Figure 6.4 shows the page displayed to users who are granted access.

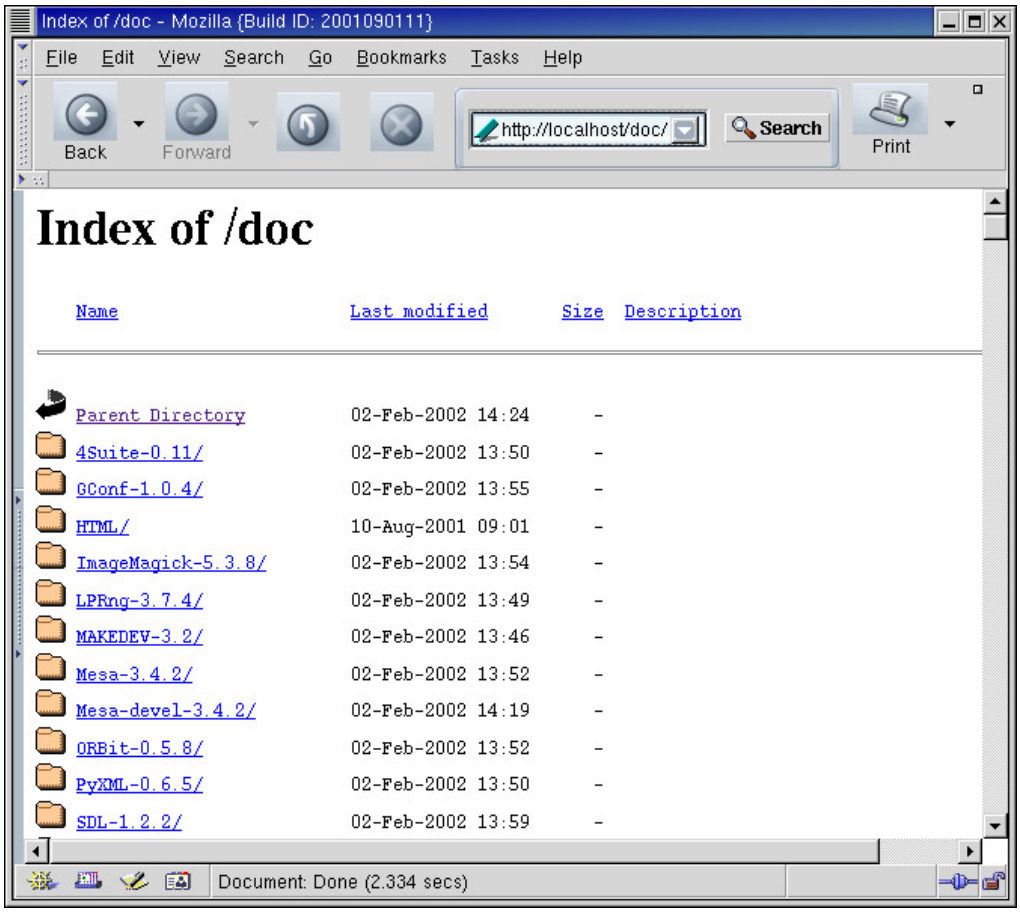


Figure 6.4: A fancy index for /usr/share/doc

Figure 6.4 shows a nice example of fancy indexing. But for this discussion, the most important thing to realize is that this index is presented only to users logged in to the localhost; that is, the web server. The specific access controls used in Listing 6.4 to limit access to users logged–on to the localhost are the following:

Order Defines the order in which the access control rules are evaluated. The order deny,allow command line tells httpd to apply the rule defined by the Deny directive first and then permit exceptions to that rule based on the rule defined by the Allow directive. The example blocks access from everyone with the deny rule, and then permit exceptions for the system that has the hostname localhost and for systems that are part of the localdomain domain with the allow rule.

Deny from Identifies hosts that are not allowed to access web documents found in this directory. The host can be identified by a full or partial hostname or IP address. A domain name can be used to match all hosts in a domain. The keyword all blocks all hosts, which is what is done in Listing 6.4.

Allow from Identifies hosts that are permitted to access documents. The host can be identified by a full or partial hostname or IP address. A domain name can be used to match all hosts in a domain. The keyword all permits all hosts, which is what was done in most of the Directory containers in Listing 6.3. After all, you usually create web data to share it with the world. However, the container in Listing 6.4 is different—it limits access to a specific host. The Allow from directive in Listing 6.4 contains both a hostname and a domain name. In Chapter 4, we saw that the hostname localhost maps to the loopback address 127.0.0.1 assigned to the internal loopback interface lo0. No external host can access the loopback interface. Further,

we saw that Red Hat creates a domain called localdomain that contains only one hostname: localhost. The Allow from directive in Listing 6.4 permits only the local host to access the /usr/share/doc directory.

Assume that you wanted to make the /usr/share/doc directory available to every host in the foobirds.org domain. Edit the httpd.conf file, changing the Allow from directive in the /usr/ share/doc container to the following:

```
allow from localhost .foobirds.org
```

This permits access through the loopback interface from the local host, and through the network from every host in the foobirds.org domain.

The example in Listing 6.4 controls access at the host level. This type of control is commonly used to segregate information for internal users from information for external customers. It is also possible to control file access at the user and group level.

Requiring User Authentication

User authentication controls file access at the user and group level by requiring a username and password before access is granted. Listing 6.5 is an example Directory container with these added levels of access control.

Listing 6.5: User Authentication for Web Access

```
<Directory /home/httpd/internal/accounting>
AuthName "Accounting"
AuthType Basic
AuthUserFile /usr/local/etc/http.passwords
AuthGroupFile /usr/local/etc/http.groups
require hdqtrs rec bill pay
order deny,allow
deny from all
allow from foobirds.org
</Directory>
```

The first two directives in this directory container are AuthName and AuthType. AuthName defines the name of the authentication realm—a value that is placed on the WWW–Authenticate header sent to the client. A *realm* is a group of server resources that share the same authentication. In the example, the directory /home/httpd/internal/accounting is the only item in the Accounting realm. But it would be possible to have other password–protected directories or documents in the Accounting realm. If we did, a user that authenticated for any resource in the Accounting realm would be authenticated for all resources in that realm.

The AuthType directive specifies the type of password authentication that will be used. This can be either Basic or Digest. When Basic is specified, a plain clear text password is used for authentication. When Digest is specified, Message Digest 5 (MD5) is used for authentication. Digest is rarely used, partly because it is not completely implemented in all browsers. But more importantly, it is not used because data that requires strong authentication is better protected using Secure Sockets Layer (SSL) security, which is covered later in this chapter.

In Listing 6.5, access is granted if the user belongs to a valid group and has a valid password. These groups and passwords are not the groups and passwords used by login. These groups and

passwords are specifically defined for the web server. The files you create for this purpose are the ones pointed to by the AuthUserFile and AuthGroupFile entries.

Add a password to the web server password file with the htpasswd command that comes with the Apache system. Add groups to the group file by editing the file with any text editor. The entries in the group file start with the group name followed by a colon and a list of users that belong to the group; for example, hdqtrs: amanda pat craig kathy.

The Require directive requires the user to enter the web username and password. In Listing 6.5, access is limited to users who belong to one of the groups hdqtrs, rec, bill, or pay; and who enter a valid password. Alternatively, the keyword valid−user could have been used on the Require directive command−line instead of a list of groups. If it were, any user with a valid password would have been given access, and the group file would have been ignored.

The Order, Deny, and Allow directives perform the same function in Listing 6.5 as they did in Listing 6.4. Listing 6.5 adds password authentication to host authentication. However, host authentication is not a prerequisite for password authentication. If the Order, Deny, and Allow directives were not used in Listing 6.5, any system on the Internet would be allowed to access the documents if the user on that system had the correct username and password.

High−Performance User Authentication

If the server has more than a few users who are required to use password authentication to access the website, the performance of the standard password file will be inadequate. The standard authentication module, mod_auth, uses a flat file that must be searched sequentially to find the user's password. Searching a flat file of only a few hundred entries can be very time consuming.

An alternative is to store the passwords in an indexed database. Two modules, mod_auth_dbm and mod_auth_db, provide support for password databases. They are used in exactly the same way as the standard flat file authentication. The only differences are the directives used to define the database inside the httpd.conf file and the command used to add passwords to the password database. The AuthUserFile directive used for the flat file is replaced by AuthDBUserFile for mod_auth_db or by AuthDBMUserFile for mod_auth_dbm. Our sample Red Hat system has the mod_auth_db module installed. Listing 6.6 shows the example from Listing 6.5 rewritten to use a database file on the sample Red Hat system.

Listing 6.6: Using *mod_auth_db* for User Authentication

```
<Directory /home/httpd/internal/accounting>
AuthName "Accounting"
AuthType Basic
AuthDBUserFile /usr/local/etc/http.passwords
AuthDBGroupFile /usr/local/etc/http.groups
require hdqtrs rec bill pay
order deny,allow
deny from all
allow from foobirds.org
</Directory>
```

The htpasswd command cannot be used to add passwords to a database file. Instead, use the command dbmmanage. The format of the dbmmanage command is:

```
dbmmanage file command username password
```

file is the filename of the database. Usernames and passwords are exactly the contents you would expect for a password database. The *command* is the keyword that provides directions to the dbmmanage command. The valid *command* values are the following:

add Adds a username and password to the database. The provided password must already be encrypted.

adduser Adds a username and password to the database. The password is provided as plain text, and is encrypted by dbmmanage.

check Checks to see if the username is in the database, and if the passwords match.

delete Removes an entry from the database.

import Copies *username*:*password* entries from stdin to the database. The passwords must already be encrypted.

update Changes the password for a user already in the database.

view Displays the contents of the database.

To add the users sara and jay to the password database, enter the commands shown in Listing 6.7.

Listing 6.7: Adding Users with *dbmmanage*

```
# cd /usr/local/etc
# dbmmanage http.password adduser sara
New password:
Re-type new password:
User sara added with password encrypted to 9jwUHif5Eu/M2
# dbmmanage http.password adduser jay
New password:
Re-type new password:
User jay added with password encrypted to MoiefJuxcM.OY
# dbmmanage http.password view
jay:MoiefJuxcM.OY
sara:9jwUHif5Eu/M2
```

Using an authentication database provides very dramatic performance improvements. Always use this feature when using user authentication for a large group of users.

Configuring SSL

The security features described previously are all designed to protect information provided by the server. In addition to protecting the security of server data, you are responsible for protecting the security of your client's data. If you want to run an electronic commerce business, you must use a secure server that protects your customers' personal information, such as credit card numbers. Secure Apache servers use Secure Sockets Layer (SSL) to encrypt protected sessions.

SSL is both more powerful and more complex than the security features discussed so far. It is more powerful because it uses public key cryptography for strong authentication and to negotiate session

encryption. When SSL is used, the exchange of data between the client and server is encrypted. Everything sent from the client and the server is protected. SSL is also more complex because external tools must be used to create the keys needed for encryption.

The mod_ssl package adds SSL support to Apache. In turn, mod_ssl depends on OpenSSL for encryption libraries, tools, and the underlying SSL protocols. Many Linux systems include OpenSSL. Before installing mod_ssl, make sure OpenSSL is installed on your system. If your distribution doesn't include OpenSSL, download the source code from http://www.openssl.org/. Run the config utility that comes with the source code and then run make to compile OpenSSL. Run make test, and make install to verify and install it.

After OpenSSL is installed, mod_ssl can be installed. Many Linux systems, including our sample Red Hat system, provide mod_ssl as part of the basic Apache system. If your distribution doesn't, download the mod_ssl package from http://www.modssl.org/. Recompile Apache using the −−with−ssl option to incorporate the SSL extensions into Apache.

The mod_ssl installation inserts various SSL configuration lines into the sample Apache configuration, usually called httpd.conf.default. These new lines are placed inside of IfDefine containers so that SSL support is an option that can be invoked from the httpd command line. Red Hat, which bundles mod_ssl into the basic system, is a good example of how this is done. Here are the IfDefine containers for the mod_ssl LoadModule and AddModule directives from a Red Hat Linux 7.2 system:

```
<IfDefine HAVE_SSL>
LoadModule ssl_module         modules/libssl.so
</IfDefine>
<IfDefine HAVE_SSL>
AddModule mod_ssl.c
</IfDefine>
```

The LoadModule and AddModule directives are only used if HAVE_SSL is defined on the httpd command line. The string HAVE_SSL is arbitrary. On another system, the string might be "SSL". The key is not what the string contains, but that it matches a value defined on the httpd command line. For example:

```
# httpd -DHAVE_SSL &
```

This command starts an SSL Apache server on a Red Hat Linux 7.2 system.

In addition to the containers for the LoadModule and AddModule directives, there are IfDefine containers that define the SSL server configuration. The active directives in the containers from the Red Hat httpd.conf file are shown in Listing 6.8. The Red Hat file contains many additional comment lines that are not shown in the listing.

Listing 6.8: Red Hat's SSL Apache Server Configuration

```
<IfDefine HAVE_SSL>
Listen 80
Listen 443
</IfDefine>
<IfDefine HAVE_SSL>
AddType application/x-x509-ca-cert .crt
AddType application/x-pkcs7-crl    .crl
</IfDefine>
<IfDefine HAVE_SSL>
```

```
<VirtualHost _default_:443>
ErrorLog logs/error_log
TransferLog logs/access_log
SSLEngine on
SSLCertificateFile /etc/httpd/conf/ssl.crt/server.crt
SSLCertificateKeyFile /etc/httpd/conf/ssl.key/server.key
<Files ~ "\.(cgi|shtml|phtml|php3?)$">
    SSLOptions +StdEnvVars
</Files>
<Directory "/var/www/cgi-bin">
    SSLOptions +StdEnvVars
</Directory>
SetEnvIf User-Agent ".*MSIE.*" \
         nokeepalive ssl-unclean-shutdown \
         downgrade-1.0 force-response-1.0
CustomLog logs/ssl_request_log \
          "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"
</VirtualHost>
</IfDefine>
```

The first IfDefine container tells the server to listen to port 443, as well as to the standard port 80. Port 443 is the standard port used by SSL.

The second IfDefine container defines two MIME file types. The file extension .crt is mapped to the MIME type application/x-x509-ca-cert, and the filename extension .crl is mapped to application/x-pkcs7-crl. These file types identify certificates and certificate revocation lists. (More about public-key certificates in a minute.)

The bulk of the SSL server configuration is contained in a VirtualHost container enclosed in the third IfDefine container. This virtual host configuration is invoked when a connection comes into the default server on port 443—the SSL port. Three special log files, two near the beginning of the container and one at the end, are created to track SSL errors and requests. (Logging is covered in more detail later.) Although no special document root is defined in Listing 6.8, most administrators insert a DocumentRoot directive into the VirtualHost container to define a directory in which secure documents are stored. Any standard configuration command can be used to configure the SSL virtual host, and there are several more directives that are valid only when SSL is running. The directives in Listing 6.8 that apply specifically to SSL are

SSLEngine Turns on SSL processing for this virtual host.

SSLOptions Sets special SSL protocol options. In the example, StdEnvVars are enabled for all files with the extensions .cgi, .shtml, .phtml, and .php3; and for the /var/www/cgi-bin directory. StdEnvVars are environment variables that are sent over the connection to the client. Retrieving these variables is time-consuming for the server, so they are sent only when it is possible that the client could use them, as is the case when CGI scripts or SSI files are involved.

ssl-unclean-shutdown In this case, the SetEnvIf directive performs essentially the same function as the BrowserMatch directives by checking to see whether the User-Agent (the browser) is Microsoft Internet Explorer. If it is, the ssl-unclean-shutdown option lets Apache know that this browser will not properly shut down the connection, and that keepalives should not be used with Internet Explorer.

SSLCertificateFile Points to the file that contains the server's public key.

SSLCertificateKeyFile Points to the file that contains the server's private key.

Public key cryptography requires two encryption keys: a public key made available to all clients, and a private key that is kept secret. The public key is in a special format called a *certificate*. Before starting SSL on your server, create these two keys.

OpenSSL provides the tools to create the public and private keys required for SSL. The simplest of these is the Makefile found in the ssl/certs directory, which allows you to create certificates and keys with a make command. Two different types of arguments can be used with the make command to create an SSL certificate or key. One type of argument uses the file extension to determine the type of certificate or key created:

make *name*.key Creates a private key and stores it in the file *name*.key.

make *name*.crt Creates a certificate containing a public key, and stores it in the file named *name*.crt.

make *name*.pem Creates a certificate and a key in the Privacy Enhanced Mail (PEM) format, and stores it in the file named *name*.pem.

make *name*.csr Creates a certificate signature request. A certificate can be digitally signed by a trusted agent, called a *certificate authority* (CA), which vouches for the authenticity of the public key contained in the certificate. (More about this later.)

Keywords are the other type of argument that can be used with this Makefile. The keywords create certificates and keys that are solely intended for use with Apache:

make genkey Creates a private key for the Apache server. The key is stored in the file pointed to by the KEY variable in the Makefile.

make certreq Creates a certificate signature request for the Apache server. The certificate signature request is stored in the file pointed to by the CSR variable in the Makefile.

make testcert Creates a certificate for the Apache server. This certificate can be used to boot and test the SSL server. However, the certificate is not signed by a recognized CA and therefore would not be acceptable for use on the Internet. The certificate is stored in the file pointed to by the CRT Makefile variable.

The /etc/httpd/conf directory on the Red Hat system has a link to the Makefile to make it easy to build the keys in the same place where the httpd.conf file expects to find them. A look at the /etc/httpd/conf directory on a Red Hat Linux 7.2 system shows that the keys pointed to by the SSLCertificateFile directive and the SSLCertificateKeyFile directive already exist, even though you did not create them.

The Makefile uses the openssl command to create the certificates and keys. The openssl command has a large and complex syntax, so the Makefile provides real benefit. However, you can use the openssl command directly to do things that are not available through the Makefile. For example, to look at the contents of the certificate that Red Hat has placed in the /etc/httpd/conf directory, enter the openssl command shown in Listing 6.9.

Listing 6.9: Examining a Certificate with the *openssl* Command

```
# openssl x509 -noout -text -in ssl.crt/server.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 0 (0x0)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=--, ST=SomeState, L=SomeCity, O=SomeOrganization,
                OU=SomeOrganizationalUnit,
                CN=localhost.localdomain/Email=root@localhost.localdomain
        Validity
            Not Before: Jul 27 12:58:42 2001 GMT
            Not After : Jul 27 12:58:42 2002 GMT
        Subject: C=--, ST=SomeState, L=SomeCity, O=SomeOrganization,
                 OU=SomeOrganizationalUnit,
                 CN=localhost.localdomain/Email=root@localhost.localdomain
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:a3:e7:ef:ba:71:2a:52:ff:d9:df:da:94:75:59:
                    07:f9:49:4b:1c:d0:67:b2:da:bd:7b:0b:64:63:93:
                    50:3d:a1:02:e3:05:3b:8e:e6:25:06:a3:d2:0f:75:
                    0a:85:71:66:d0:ce:f9:8b:b0:73:2f:fe:90:75:ad:
                    d6:28:77:b0:27:54:81:ce:3b:88:38:88:e7:eb:d6:
                    e9:a0:dd:26:79:aa:43:31:29:08:fe:f8:fa:90:d9:
                    90:ed:80:96:91:53:9d:88:a4:24:0a:d0:21:7d:5d:
                    53:9f:77:a1:2b:4f:62:26:13:57:7f:de:9b:40:33:
                    c3:9c:33:d4:25:1d:a3:e2:47
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                55:E9:ED:C1:BF:1A:D4:F8:C2:78:6E:7A:2C:D4:9C:AC:7B:CD:D2
            X509v3 Authority Key Identifier:
                keyid:55:E9:ED:C1:BF:1A:D4:6E:7A:2C:D4:DD:9C:AC:7B:CD:D2
                DirName:/C=-/ST=SomeState/L=SomeCity/O=SomeOrganization/
                       OU=SomeOrganizationalUnit/CN=localhost.localdomain/
                       Email=root@localhost.localdomain
                serial:00
            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: md5WithRSAEncryption
        76:78:77:f0:a2:19:3b:39:5f:2a:bd:d0:42:da:85:6e:c2:0c:
        5e:80:40:9c:a8:65:da:bf:38:2b:f0:d6:aa:30:72:fb:d3:1d:
        ce:cd:19:22:fb:b3:cc:07:ce:cc:9b:b6:38:02:7a:21:72:7c:
        26:07:cc:c9:e0:36:4f:2f:23:c9:08:f7:d4:c1:57:2f:3e:5c:
        d5:74:70:c6:02:df:1a:62:72:97:74:0a:a6:db:e0:9d:c9:3d:
        8e:6b:18:b1:88:93:68:48:c3:a3:27:99:67:6f:f7:89:09:52:
        3a:a3:fb:20:52:b0:03:06:22:dd:2f:d2:46:4e:42:f2:1c:f0:
        f1:1a
```

There is a lot of information in a certificate. But only a few pieces of it are needed to determine whether or not this is a valid certificate for our server:

Issuer The Issuer is the *distinguished name* of the CA that issued and signed this certificate. A distinguished name is a name format designed to uniquely identify an organization. It is clear in the example that the name of the Issuer is just an example, not a real organization.

Subject The Subject is the distinguished name of the organization to which this

certificate was issued. In our case, it should be the name of our organization. Clearly, the Subject in this certificate is just a sample.

Validity The Validity is the timeframe in which this certificate is valid. In the example, the certificate is valid for a year. Because the dates are valid, this certifi–cate can be used to test SSL.

To test that the SSL server is indeed running, use a browser to attach to the local server. However, instead of starting the URL with http://, start it with https://. https connects to port 443, which is the SSL port. The browser responds by warning you that the server has an invalid certificate, as shown in Figure 6.5.

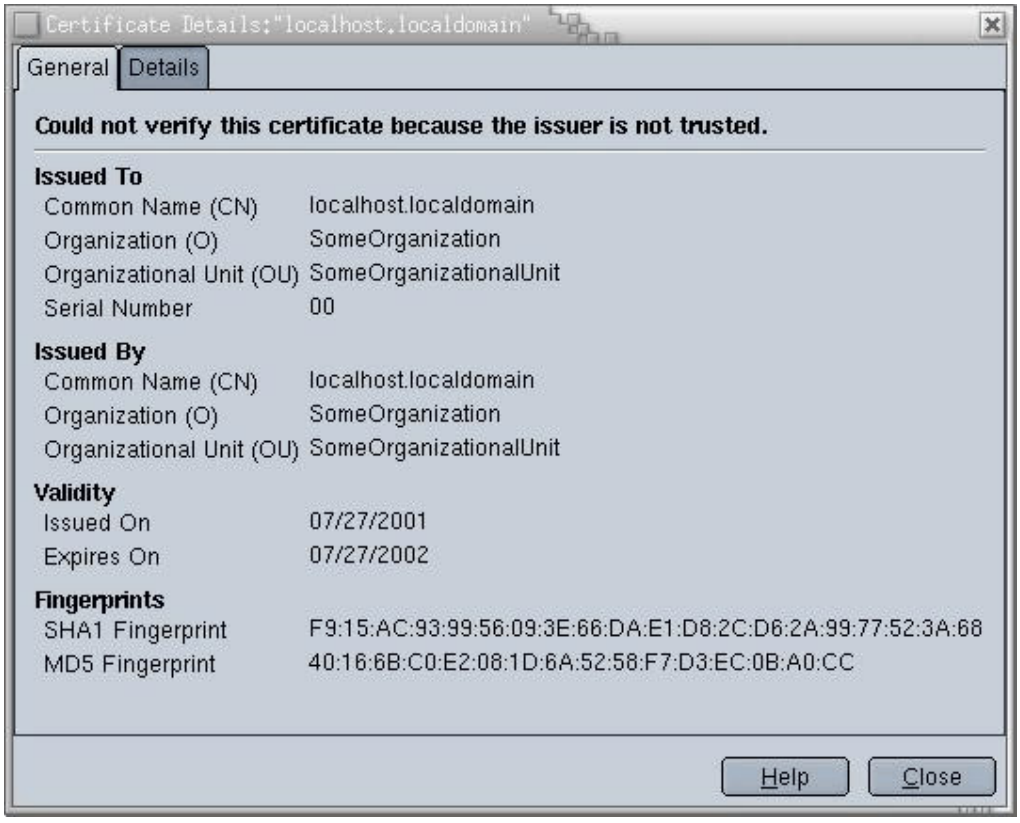


Figure 6.5: An invalid certificate warning

Clicking on View shows some of the same certificate information we saw in Listing 6.9. You can accept the certificate for this session, and connect to the "secure document." In this case, the secure document is just a test page because we have not yet stored any real secure documents on the system.

The server is up and running, but it cannot really be used by external customers until the certificate has a valid signature. Use make certreq to create a certificate signature request (CSR) specific to your server, as shown in Listing 6.10.

Listing 6.10: Creating an Apache Certificate Signature Request

```
# cd /etc/httpd/conf
# make certreq
umask 77 ; \
/usr/bin/openssl req -new -key /etc/httpd/conf/
ssl.key/server.key -out /etc/httpd/conf/ssl.csr/server.csr
Using configuration from /usr/share/ssl/openssl.cnf
You are about to be asked to enter information that will be
```

```
incorporated into your certificate request.
What you are about to enter is what is called a
Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:Maryland
Locality Name (eg, city) []:Gaithersburg
Organization Name (eg, company) [Internet Widgits Ltd]:foobirds.org
Organizational Unit Name (eg, section) []:Headquarters
Common Name (eg, your name or hostname)[]:wren.foobirds.org
Email Address []:alana@foobirds.org

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

The freshly created request can be examined using the openssl command. Notice that Listing 6.11 shows that this request has a valid Subject containing a distinguished name that identifies the sample server. However, there is no Issuer. This request needs to be signed by a recognized CA to become a useful certificate.

Listing 6.11: Examining a Certificate Signature Request with *openssl*

```
# openssl req -noout -text -in server.csr
Using configuration from /usr/share/ssl/openssl.cnf
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=US, ST=Maryland, L=Gaithersburg, O=foobirds.org,
                 OU=Headquarters,
                 CN=wren.foobirds.org/Email=alana@foobirds.org
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:a3:e7:ef:ba:71:2a:52:ff:d9:df:da:94:75:59:
                    07:f9:49:4b:1c:d0:67:b2:da:bd:7b:0b:64:63:93:
                    50:3d:a1:02:e3:05:3b:8e:e6:25:06:a3:d2:0f:75:
                    0a:85:71:66:d0:ce:f9:8b:b0:73:2f:fe:90:75:ad:
                    d6:28:77:b0:27:54:81:ce:3b:88:38:88:e7:eb:d6:
                    e9:a0:dd:26:79:aa:43:31:29:08:fe:f8:fa:90:d9:
                    90:ed:80:96:91:53:9d:88:a4:24:0a:d0:21:7d:5d:
                    53:9f:77:a1:2b:4f:62:26:13:57:7f:de:9b:40:33:
                    c3:9c:33:d4:25:1d:a3:e2:47
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: md5WithRSAEncryption
        3f:c2:34:c1:1f:21:d7:93:5b:c0:90:c5:c9:5d:10:cd:68:1c:
        7d:90:7c:6a:6a:99:2f:f8:51:51:69:9b:a4:6c:80:b9:02:91:
        f7:bd:29:5e:a6:4d:a7:fc:c2:e2:39:45:1d:6a:36:1f:91:93:
        77:5b:51:ad:59:e1:75:63:4e:84:7b:be:1d:ae:cb:52:1a:7c:
        90:e3:76:76:1e:52:fa:b9:86:ab:59:b7:17:08:68:26:e6:d4:
        ef:e6:17:30:b6:1c:95:c9:fc:bf:21:ec:63:81:be:47:09:c7:
        67:fc:73:66:98:26:5e:53:ed:41:c5:97:a5:55:1d:95:8f:0b:
        22:0b
```

CAs are commercial, for–profit businesses. Fees and forms, as well as the CSR, are required to get a certificate signed. Web browsers contain a list of recognized CAs. On a Netscape 6.1 browser, view this list in the Certificate Manager in Preferences, as shown in Figure 6.6. All CAs have websites that provide the details of the cost and the application process.

Figure 6.6: The CAs built–in Netscape 6.1

Although certificates signed by a recognized CA are the most widely used, it is possible to create a self–signed certificate. But this has very limited utility. As we saw in Figure 6.5, a certificate that is not signed by a recognized CA must be manually accepted by the client. Therefore, self–signed certificates can be used only if you have a small client base. Use the openssl command to sign the certificate yourself:

```
# openssl req -x509 -key ssl.key/server.key \
> -in ssl.csr/server.csr -out ssl.crt/server.crt
```

Examining the newly created server.crt file with openssl shows that the Issuer and the Subject contained the same distinguished name. But this time, the name is the valid name of our server.

Managing Your Web Server

Despite the enormous number of directives used to configure Apache, configuration is not normally the biggest task involved in running a web server. Configuration usually requires no more than adjusting a few configuration options when the server is first installed, after which the configuration remains stable. On the other hand, monitoring the server's usage and performance and ensuring its reliability and security are daily tasks.

Monitoring Your Server

Apache provides tools to monitor the status of your server and logs that keep a history of how the system is used and how it performs over time. One of these tools is the server–status monitor. To use this monitor, it must either be compiled into httpd or installed as a dynamically loadable module. The following two lines from the Red Hat httpd.conf configuration file load the required module:

```
LoadModule status_module      modules/mod_status.so
AddModule mod_status.c
```

To get the maximum amount of information from the server–status display, add the ExtendedStatus directive to your httpd.conf file. By default, it is commented out of the Red Hat configuration. Remove the hash mark (#) to active this directive; for example:

```
ExtendedStatus on
```

Enable the monitor by locating the Location /server–status container directive in the httpd.conf file and removing the hash marks to activate the directives in that container. Edit the Allow from directive to control access to the server status screen. For example, you might grant access to the localhost or to all hosts in your domain. Listing 6.12 shows the uncommented container after it has been configured to allow access from all hosts in the foobirds.org domain.

Listing 6.12: The Server–Status Location Container

```
<Location /server-status>
SetHandler server-status
order deny,allow
deny from all
allow from foobirds.org
</Location>
```

After the monitor is installed and enabled, access it from a browser at www.foobirds.org/server–status/?refresh=20. The refresh value is not required, but when used, the status display is automatically updated. In Listing 6.12, we ask for a status update every 20 seconds. Figure 6.7 shows the status screen for our test server.

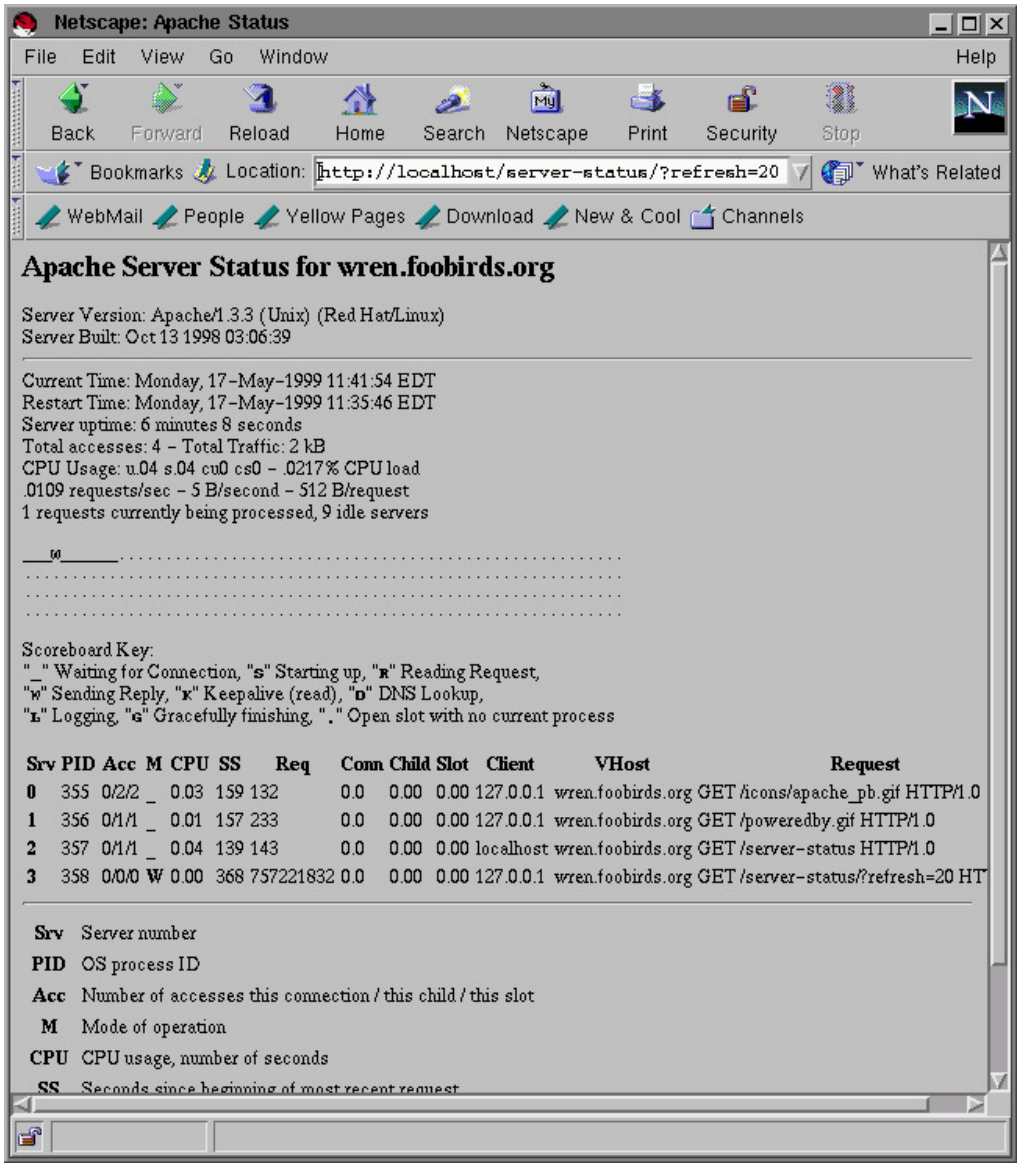


Figure 6.7: The Apache server–status display

Monitoring tells you about the real–time status of your server, but even more can be learned by looking at the way your server is used over time. Logging provides that information.

Apache Logging

Several directives in the Red Hat httpd.conf file configure logging. The ErrorLog directive defines the path of the error log file. Use the error log to detect failures. Review the error log at least once a day and look for problems. To keep a close eye on the file while you're logged–in, use the tail command with the –f option:

```
$ tail -l 1 -f /var/log/httpd/apache/error_log
```

The tail –l 1 command prints the last record in the error file, and the –f option keeps the tail process running so that you will see any record as it is written to the file. This allows you to monitor the file in real time.

The LogLevel directive defines what types of events are written to the error log. The Red Hat configuration specifies that warnings and other more critical errors are to be written to the log. There

are eight possible LogLevel settings: debug, info, notice, warn, error, crit, alert, and emerg. The log levels are cumulative. debug provides debugging information and all other types of logging. warn provides warnings, errors, critical messages, alerts, and emergency messages. debug causes the file to grow at a very rapid rate. emerg keeps the file small, but only notifies you of disasters. warn is a good compromise between enough detail and too much detail.

The TransferLog directive defines the path to the log in which httpd writes information about server activity. Just as important as errors, the logs provide information about who is using your server, how much it is being used, and how well it is servicing the users. Use the transfer log to monitor activity and performance. Web servers are used to distribute information. If no one wants or uses the information, you need to know it. Much of the logging configuration controls what activity is logged, as well as where it is logged.

The LogFormat directives define the format of log file entries. The files that these entries are written to are defined by the CustomLog directives. In the default Red Hat configuration, there are four active LogFormat directives and one active CustomLog directive:

```
CustomLog logs/access_log combined
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\"
Ä\"%{User-Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
```

Notice that a CustomLog statement and an associated LogFormat statement end with the same label. This label is an arbitrary name used to bind the format and the file together. In the example, the LogFormat that ends with the label "combined" is the format linked to the CustomLog directive. The LogFormat directives have a complex syntax.

Defining Log Formats

Apache log files conform to the Common Log Format (CLF). CLF is a standard used by all web server vendors. Using this format means that the logs generated by Apache servers can be processed by any log–analysis tool that also conforms to the standard, and most do.

The format of a standard CLF entry is defined by the following LogFormat directive from our sample httpd.conf file:

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
```

A CLF entry contains seven fields, each represented by a parameter in the LogFormat directive:

> **%h** Logs the IP address or hostname of the client. If HostnameLookups is set to on, this is the client's fully qualified hostname. On the sample Red Hat system, this would be the IP address because HostnameLookups is turned off to enhance server performance.
>
> **%l** Logs the username assigned to the user on the client. The username is retrieved using the identd protocol. Most clients do not run identd, and thus do not provide this information, so this field usually contains a hyphen to indicate a missing value
>
> **%u** Logs the username used to access a password protected web page. This should match a name defined in the AuthUser file or the AuthDBMUser database you created for the server. Most documents are not password–protected; therefore, in

most log entries, this field contains a hyphen.

%t Logs the date and time.

%r Logs the first line of the request, which is often the URL of the requested document. The \" characters are just there to insert quotes in the output.

%>s Logs the status of the last request. This is the three−digit response code that the server returned to the client. (More on response codes in a minute.) The > is a literal character that will appear in the log file in front of the response code.

%b Logs the number of bytes sent.

The format of the LogFormat directive is enclosed in quotes. The label "common" is not part of the format. It is an arbitrary string used to tie the LogFormat directive to a CustomLog directive. In the default Red Hat configuration, this particular LogFormat directive is not used by a CustomLog directive. Instead, the Red Hat configuration uses the following "combined" LogFormat.

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\"
Ä\"%{User-Agent}i\"" combined
```

Notice that this LogFormat starts with the same seven parameters as the "common" format to which it adds more information. Apache logs can be customized to log just the information you want to track.

In addition to the standard CLF fields, Apache can log the contents of any header records received or sent. For example, to log the value received from the client in the User−agent header, add the following to a LogFormat directive:

```
%{User-agent)i
```

This works for any header. Simply replace User−agent with the name of the header. The i indicates this is an input header. To log an output header, use an o at the end of the description.

The "combined" LogFormat used on our sample Red Hat system logs everything in the CLF plus the contents of the input User−agent and Referer headers. The User−agent header contains the name of the browser used by the client. The Referer header contains the name of the remote server that linked to your web page.

Using Conditional Logging

Apache also supports conditional logging, which logs specified fields only when certain conditions are met. The conditions that can be tested for are the status codes returned by the server. The status codes are

200: OK The request is valid.

302: Found The requested document was found.

304: Not Modified The requested document has not been modified.

400: Bad Request The request is invalid.

401: Unauthorized The client or user is denied access.

403: Forbidden The requested access is not allowed.

404: Not Found The requested document does not exist.

500: Server Error There was an unspecified server error.

503: Out of Resources (Service Unavailable) The server has insufficient resources to honor the request.

501: Not Implemented The requested server feature is not available.

502: Bad Gateway The client specified an invalid gateway.

To make a field conditional, put a status code on the field in the LogFormat entry. For example, suppose you want to log the browser name only if the browser requests a service that is not implemented in your server. Combine the Not Implemented (501) status code with User–agent header in this manner:

```
%501{User-agent}i
```

If this value appears in the LogFormat, the name of the browser is only logged when the status code is 501.

You can also use an exclamation point to specify that you want to log a value only when the status code is not a certain value; the exclamation point indicates "not." For example, to log the address of the site that referred the user to your web page if the status code is not one of the good status codes, add the following to a LogFormat:

```
%!200,302,304{Referer}i
```

This particular conditional log entry is actually very useful. It tells you when a remote page has a stale link pointing to your website. It also shows that you can use multiple status codes. Use the LogFormat directive to define exactly what information is logged and the conditions under which it is logged.

In Sum

Web servers are an essential part of any organization's Internet. Linux is an excellent platform for a web server using the Apache software that is included in the distribution. Apache is the most popular web server on the Internet. With Linux, it can effectively support a large organization's website.

Thus far, this book has covered some of the services, such as e–mail, DNS, and the Web, that Linux does best. The next chapter concludes the part of this book that focuses on creating an Internet server with a look at how Linux can be used to create a low–cost Internet router. Although Linux may not create the most powerful router, it is certainly one of the most cost–effective.

Chapter 7: Network Gateway Services

Overview

A computer can communicate directly only with computers with which it shares a physical connection. Given this fact, the computer on your desk should be able to communicate only with computers that are electrically connected to the network cable that connects to your system. So, how does it communicate with a computer on the other side of the world? There are two primary techniques: circuit switching and packet switching.

Circuit switching is the technique used by the voice telephone network. When you pick up the telephone, you hear a dial tone. At this point, you have an electrical connection to the telephone switch at the local telephone company's central office. As you dial the telephone number, you provide the switch with the information it needs to make additional connections. Using this information, the switch connects your inbound port to an outbound port. If the number you are calling is serviced by the local switch, it sets up a connection between the port your telephone is attached to and the port connected to the phone you're calling. If the number you are calling is remotely located, the local switch sets up a connection to the next switch down the line. Each switch connects to the next switch in line until the switch servicing the remote phone is reached. This creates a circuit from your phone to the remote phone, wherever it is located, which is dedicated to your use until you hang up the phone. When your computer communicates over a modem, it uses the telephone system to create a circuit between itself and the remote system, which is often the server at an ISP that connects your system into the packet–switched Internet.

Packet switching is the technique used by most data networks. Every packet in the network contains an address that tells the switch where the packet is bound. When the packet arrives at a switch, the switch reads the address and determines how the packet should be forwarded. If the switch has a physical connection to the destination node, it delivers the packet itself. Otherwise, it forwards the packet to the next switch in the path toward the destination node. Each packet is handled separately. No end–to–end connection is established

In the circuit–switched model, the connection is between your phone and the phone at the remote end. In the packet–switched system, the connection is between your host and the local router. Figure 7.1 illustrates that packet switches use hop–by–hop routes versus the end–to–end connections used by circuit switches.

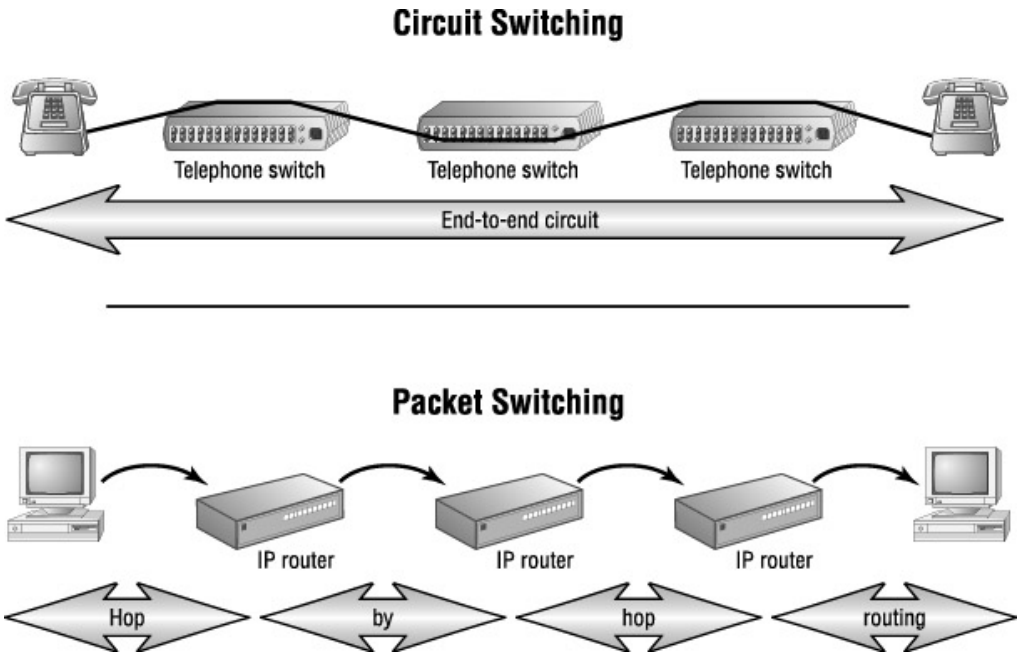


Figure 7.1: Circuit switching versus packet switching

Note Whether you configure your system as a host or as an IP router, it will not have end–to–end knowledge of the routes through the network. It will know only about local routers.

The Internet—and all TCP/IP networks—are packet–switched networks. An IP packet switch is called a *gateway* or an *IP router*. Routers interconnect networks, moving data from one network to another until the destination network is reached. At that point, direct delivery is made to the destination host. This is illustrated in Figure 7.2.

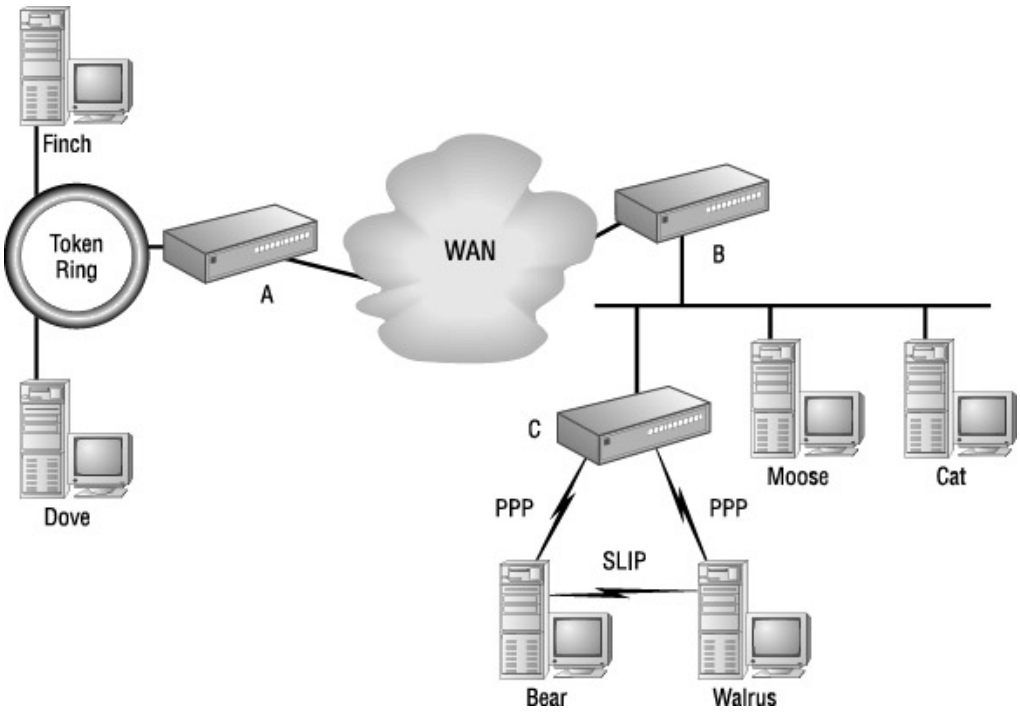


Figure 7.2: Routing through networks

In the figure, an IP datagram from finch to walrus would first go to router A, then to router B, then to router C, and finally to walrus. Notice that IP routers can interconnect different types of physical networks. And, as this chapter demonstrates, any Linux system can be configured to be an IP router.

Understanding Routing

Routing turns TCP/IP networks into an internet and is an essential function of the Internet Protocol (IP). Even a Linux system, which has the Transport layer and the Application layer sitting above the IP layer, makes routing decisions in the IP layer. When the IP layer receives a packet, it evaluates the destination address in the header of the packet, as follows:

- If the destination address is the address of the local computer, IP evaluates the protocol number and passes the packet up to the appropriate transport protocol. (See the discussion of protocol numbers in Chapter 3, "Login Services.")
- If the destination address is on a directly connected network, IP delivers the packet to the destination host.
- If the destination is on a remote network, IP forwards the datagram to a local router. The router that the packet is sent to must share a physical network with the local system. It is the responsibility of that router to then forward the packet on to the next router and so on, hop by hop, until the packet reaches its destination.

Based on this list of possible decisions, IP will either directly deliver the packet or forward it to a router for additional processing. From reading Chapter 3, you know how IP uses the protocol number and the port number to deliver data to the correct application within the local host. But you may not yet know how IP delivers data across a network. To deliver a packet to another host on a directly attached network, IP must use the Physical layer addressing of that network by converting the IP address to a Physical layer address.

Converting IP Addresses to Ethernet Addresses

As Figure 7.2 illustrates, IP can run over many different types of networks. The IP address is a logical address. The address means something to the logical IP network, but it doesn't mean anything to the physical networks over which IP must transport the data. To send data over a physical network, IP must convert the IP address to an address understood by the network. The most common example of this is the conversion from an IP address to an Ethernet address. The protocol that performs this conversion is the Address Resolution Protocol (ARP).

Note The ARP protocol is included with Linux systems, and is installed by default as part of the TCP/IP network software. You don't have to do anything to activate ARP, and it should run without problems.

The ARP protocol dynamically builds a table that maps IP addresses to Ethernet addresses. It does this using the broadcast facilities of the Ethernet. When ARP receives a request to convert an IP address to an Ethernet address, it checks to see whether it has the mapping for that address in the ARP table. If the mapping is there, it uses it. If it isn't in the table, ARP sends a broadcast on the Ethernet, asking who owns the IP address. When a computer sees an ARP broadcast for its IP address, it responds with its Ethernet address. ARP then adds that response to the table.

Use the arp command to examine the contents of the ARP table on your Linux system. Use the –a command–line option to view the entire table, as shown in Listing 7.1.

Listing 7.1: Viewing the *arp* Cache

```
$ arp -a
bluejay (172.16.55.1) at 00:00:C0:4F:3E:DD [ether] on eth0
duck (172.16.55.11) at 00:10:4B:87:D4:A8 [ether] on eth0
```

```
raven (172.16.55.251) at 08:00:20:82:D5:1D [ether] on eth0
gw50 (172.16.50.254) at 00:00:0C:43:8D:FB [ether] on eth1
```

The arp command lists the hostname, IP address, and Ethernet address of every system currently stored in the ARP table. The keyword ether enclosed in square brackets indicates the hardware type. This will always be ether on an Ethernet network. There are other hardware values, but they are for obscure networks such as ARCnet. Most of the entries in the example are for the network connected to network interface eth0.

However, a router has more than one network interface, so it is possible to see an arp display with more than one Ethernet interface indicated. The last line in the display shows this. gw50 is reached through interface eth1.

You can also use the arp command to check for the table entry of an individual host:

Listing 7.2: Viewing a Single *arp* Table Entry

```
$ arp bluejay
Address  HWtype  HWaddress          Flags Mask   Iface
bluejay  ether   00:00:C0:4F:3E:DD  C            eth0
```

Note It is possible to enter the arp command looking for a specific host and to receive the response no entry. This does not necessarily indicate a problem. Try sending a ping to the host first in order to prime the cache. Then, enter the arp command. You should see the correct table entry.

Listing 7.2 contains much the same information as Listing 7.1. Again, there's the hardware type, the Ethernet address, the network interface name, and the hostname. (If you prefer the IP address to the hostname, use the –n option on the arp command line.) There are two fields, however, that you didn't see in the earlier listing: the Flags field and the Mask field.

The Flags field can contain three possible values:

C Indicates that this is a complete entry. To be valid, an entry must be complete. Therefore, the C flag should always be set.

M Indicates a static entry that was manually entered. ARP table entries are normally dynamic. They are learned from the computers on the network, and they are held in the ARP table for only a few minutes. However, the system administrator can place static entries in the table. These entries stay in the table as long as the system is running. See Chapter 13, "Troubleshooting," for information on how a static entry is used to troubleshoot an address assignment problem.

P Indicates an entry that will be *published*. In other words, if this computer receives an ARP broadcast for the IP address in this entry, the local host responds with the Ethernet address, even though the IP address does not really belong to the local host. This is called proxy ARP, which is used to help systems that cannot respond themselves. See the "Proxy ARP" sidebar for an example of when this is used.

The Mask field contains an optional network mask, if one is used. By default, the mask is 255.255.255.255, which says that the entire IP address is matched to the Ethernet address. Other masks are rarely used. Solaris Unix systems use the mask 240.0.0.0 to map multicast addresses to

the Ethernet broadcast address. Sometimes, an optional mask value is used to publish a single Ethernet address for an entire subnet. In that case, the mask required for the specific subnet is used. However, this is not recommended. Subnets should be connected through routing, not through proxy ARP.

Proxy ARP

ARP requests are sent via Ethernet broadcasts. It is possible for a host to connect to an Ethernet through another network technology that cannot respond to an ARP request. To address this problem, you can use proxy ARP.

Assume that two systems connect to subnet 172.16.55.0 through bluejay using some hardware that does not respond to ARP requests. Both killdeer (172.16.55.8) and meadowlark (172.16.55.23) systems have been assigned addresses on subnet 172.16.55.0. bluejay is configured to provide proxy ARP for both systems with the following commands:

```
# arp -s killdeer 00:00:C0:4F:3E:DD pub

# arp -s meadowlark 00:00:C0:4F:3E:DD pub

# arp killdeer

Address      HWtype  HWaddress           Flags Mask    Iface

killdeer     ether   00:00:C0:4F:3E:DD   CMP           eth0

# arp meadowlark

Address      HWtype  HWaddress           Flags Mask    Iface

meadowlark   ether   00:00:C0:4F:3E:DD   CMP           eth0
```

The –s command–line argument tells arp that this is a static entry, and the pub argument says that this entry will be published. Notice that the same Ethernet address is used for both killdeer and meadowlark, and that the address is the Ethernet address of bluejay. bluejay responds to ARP requests with its own Ethernet address so that it receives packets bound for killdeer and meadowlark. Because bluejay is configured to forward packets, when it receives packets for these systems, it sends those packets to the correct host through the non–Ethernet hardware those systems use.

The IP address must be converted to a Physical layer address for all types of external data delivery, whether the system is making a direct delivery or forwarding a packet for further processing. A traditional host only accepts packets from the network that are addressed to the host. It does not accept packets addressed to other hosts or forward those packets on. Routers, on the other hand, do exactly that. To get this behavior, you must enable forwarding on a router.

Enabling IP Packet Forwarding

When a computer forwards a packet that it has received from the network on to a remote system, it is called *IP forwarding*. All Linux systems can be configured to forward IP packets. In general, hosts do not forward datagrams, but routers must.

To use a Linux system as a router, enable IP forwarding by setting the correct value in the /proc/sys/net/ipv4/ip_forward file. If the file contains a 0, forwarding is disabled. If it contains a 1, forwarding is enabled. A cat of the ip_forward file shows the current setting for your system:

```
$ cat /proc/sys/net/ipv4/ip_forward
0
```

Write a 1 to the file to enable forwarding:

```
[root]# echo "1" > /proc/sys/net/ipv4/ip_forward
[root]# cat /proc/sys/net/ipv4/ip_forward
1
```

If you intend to run your Linux system as a router, stow this command in the rc.local file to enable forwarding every time the system boots.

On Red Hat Linux 7.2, you can edit /etc/sysctl.conf and change net.ipv4.ip_forward = 0 to net.ipv4.ip_forward = 1. Other distributions may use different configuration files for the same purpose. Writing a value directly to /proc/sys/net/ipv4/ip_forward should work on all current distributions.

Regardless of whether the system is a router or a host, it can make final delivery only if it is on the same network as the destination host. In all other cases, the system must send the packet on to a router. The routing table tells the local system which router the packet should be sent to.

The Linux Routing Table

Chapter 2, "The Network Interface," described the structure of an IP address, explaining that it is composed of a network portion and a host portion. Routing is network–oriented; IP makes its decision on whether to directly deliver the packet or forward the packet to a router based on the network portion of the address. When the decision is made to forward the packet to a router, IP looks in the routing table to determine which router should handle the packet.

The Linux routing table is displayed by entering the route command with no command–line arguments. Because I prefer to look at network numbers instead of hostnames when analyzing a routing table, I use the –n option, which prevents route from converting IP addresses to hostnames for the routing table display. Listing 7.3 is an example.

Listing 7.3: A Simple Routing Table

```
$ route -n
Kernel IP routing table
Destination Gateway       Genmask       Flags Metric Ref Use Iface
172.16.55.0 0.0.0.0       255.255.255.0 U     0      0     0 eth0
172.16.50.0 172.16.55.36  255.255.255.0 UG    0      0     0 eth0
127.0.0.0   0.0.0.0       255.0.0.0     U     0      0     0 lo
0.0.0.0     172.16.55.254 0.0.0.0       UG    1      0     0 eth0
```

Note On some Unix systems (Solaris is an example), use netstat –r to list the routing table.

The routing table in Listing 7.3 contains the following fields:

> **Destination** The destination network (or host). Normally, this is a network, but as you'll see in the Flags field discussion, it is possible to have a host–specific route.

For the default route, this field contains all zeros (0.0.0.0).

Gateway The gateway for the specified destination. If this field contains all zeros (0.0.0.0) or an asterisk (*), it means that the destination network is directly connected to this computer and that the "gateway" to that network is the computer's network interface.

Genmask The bit mask applied to addresses to see if they match the destination address.

Flags The flags describe certain characteristics of this route. The possible flag values are as follows:

> **U** This route is up and operational.
>
> **H** This is a route to a specific host. None of the routes in the example are host–specific routes, which are used only for special purposes.
>
> **G** This route uses an external gateway. The system's network interfaces provide routes to directly connected networks. All other routes use external gateways. Directly connected networks do not have the G flag set; all other routes do.
>
> **D** This route was added dynamically by a routing protocol or an ICMP Redirect Message. (There is much more about routing protocols later in this chapter.) When a system learns of a route via an ICMP Redirect, it adds the route to its routing table so that additional packets bound for that destination will not need to be redirected.
>
> **R** This route was reinstated after being updated by a dynamic routing protocol. Routes can be configured as passive or static routes, even when a dynamic routing protocol is being used.
>
> **M** This route was modified by a dynamic routing protocol.
>
> **A** This route was added during address configuration. When an address is defined for a local interface using the ifconfig command, a route for that interface is added to the routing table. In practice, the A flag does not appear in the default routing table display.
>
> **C** This route is in the routing cache. Routes are temporarily cached when they are used to create a connection. The cache can be examined directly by running the route command with the –C option (for example, route –Cn). In practice, the C flag does not appear in the routing table display.
>
> **!** This route has been marked as a bad route.

Metric This is the routing cost for this interface. Normally, this field is zero for routes to directly connected networks. Routes to external gateways often have a metric of 1, though that is not always the case. The metric is an arbitrary value that you set. The higher the metric, the more "expensive"—and, therefore, the less preferred—the

route. There is more about routing metrics in the "Routing Protocols" section, later in this chapter.

Ref Shows the number of times the route has been referenced to establish a connection.

Use Shows the number of times this route was looked up in the cache. This is useful only when examining the cache; in the basic routing table display shown in Listing 7.3, this field is unused.

Iface The name of the network interface used by this route. For Ethernet network interfaces, the names will be eth0, eth1, eth2, and so on. For PPP network interfaces, the names will be ppp0, ppp1, ppp2, and so on.

With this knowledge of how the routing table is displayed, take a look at each line in the sample table:

- The first line defines the connection of this host to the local Ethernet. From this entry, you can tell that the host is connected to subnet 172.16.55.0 and that it connects to that subnet directly through interface eth0.
- The second line is a static route added by the network administrator. It tells you that 172.16.55.36 is the gateway to subnet 172.16.50.0/24.
- The third line defines the loopback network, and states that the gateway to the loopback network is the interface lo.
- The last line defines the default route, which identifies the default gateway. If the destination of a packet does not match a specific route in the routing table, the packet is sent to the default gateway. In our sample system, specific routes exist for networks 172.16.55.0, 172.16.50.0, and 127.0.0.0. All other destinations are sent to the default router.

Notice that both of the external gateways are connected to local Ethernet 172.15.55.0/24. If they weren't, the local system could not communicate with them and thus could not communicate with the outside world.

Routes enter the routing table in one of two ways: either the system administrator enters them as static routes, or they are added to the table by a routing protocol as dynamic routes. The next few sections look at both techniques for defining network routes.

Defining Static Routes

Static routes are defined on every Linux system that connects to a TCP/IP network. A minimal routing table has a route for the loopback address and a route for the network interface. Every Linux system on a TCP/IP network has at least these two static routes. These two routes from the sample routing table shown in Listing 7.3 are

```
172.16.55.0 0.0.0.0   255.255.255.0  U   0    0   0  eth0
127.0.0.0   0.0.0.0   255.0.0.0      U   0    0   0  lo
```

Under Linux kernel 2.4, these routes are created by the ifconfig commands that configure the network interfaces. Systems that use older kernels create these routes with specific route commands. For example:

```
route add -net 172.16.55.0 netmask 255.255.255.0 dev eth0
```

```
route add -net 127.0.0.0   netmask 255.0.0.0     dev lo0
```

Even on old systems, you do not need to enter these commands yourself. They are stored in one of the startup scripts that come with those systems.

The *route* Command

Additional static routes are defined by the route command. It allows anyone to display the routing table and allows the root user to add and delete routes in the table. The system administrator added this static route to the sample routing table we saw in Listing 7.3:

```
172.16.50.0 172.16.55.36 255.255.255.0 UG 0 0 8 eth0
```

The route statement that creates this route is

```
route add -net 172.16.50.0 netmask 255.255.255.0 gw 172.16.55.36
```

Examining this command shows almost everything you need to know about the route command syntax. All route commands start with an option that defines the "action" of the route command, which is either add or delete. All of the sample commands add routes to the routing table. To change a route, first delete it and then add it back in with the necessary corrections.

The –net option tells route that you are adding a network route. The alternative is –host for a host–specific route, but this is rarely used because most routes are network routes.

The –net option is followed by the destination address and by the network mask that is used to determine the network portion and the host portion of the address. The network mask must be preceded by the keyword netmask.

Warning Always define the netmask value yourself. If you don't, the address is interpreted using the natural mask, which means that the address is interpreted according to the old address class rules. Be specific. Define the mask yourself.

All of the sample route commands end with either an external gateway or a local device name. When an external gateway is used, it is defined by the gw option and the IP address of the gateway. When the interface device name is used, it is usually preceded by the keyword dev, although this is not required. The network interface name is included on a route statement that defines the connection of the device to the local network. On other route statements, the interface name is optional. Use the device name on all route statements when you have more than one interface to ensure that the route uses the interface that you intend.

Defining the Default Route

Thus far, all of the routes from the sample routing table have been discussed, except for the default route. The sample table contains a default route that specifies 172.16.55.254 as the default router:

```
0.0.0.0   172.16.55.254 0.0.0.0    UG 1 0 17 eth0
```

Enter the following command to define that route:

```
route add default gw 172.16.55.254
```

This command looks similar to the previous route command, except that the keyword default is used in place of the destination network. It is the presence of this keyword that defines the default route.

Most Linux systems connected to a TCP/IP internet have a static default route. Because it is so common, it is unlikely that you need to enter a route command to define it. All Linux installations that I have worked with ask you for the address of the default router during the initial installation. Provide it at that time. The system stores the value and then uses it to define the default route during the boot. For example, Red Hat stores the default gateway address in the /etc/sysconfig/ network file.

For most Linux servers, all you need to do is define the default route because most servers are hosts that depend on a single external router for routing service. On occasion, however, a Linux server may be used as the router for a small network. When it is, you may need to run a routing protocol, as discussed next.

Using Dynamic Routing

Static routing tables are most efficient when there are a limited number of routers. If the network has only one router, the correct configuration is to use a static default route. Use dynamic routing protocols when the routing environment of a network is changeable or complex; for example, when there are multiple routers that can reach the same destinations.

Routing Protocols

Routing and routing protocols are not the same thing. All Linux systems make routing decisions, but very few systems run routing protocols. Routing protocols perform two functions: They select the "best" route to a destination, and they communicate that route to other routers on the network. Thus a *routing protocol* is a technique for defining routes and for disseminating routes.

There are several different routing protocols, and considering the fact that Linux systems are not usually used as routers, a surprising number of these protocols are available for Linux systems. Protocols are differentiated by the metric they use for determining the best route and by the technique they use for distributing routing information.

Routing protocols are divided into interior protocols and exterior protocols. *Interior protocols* are used inside of a routing domain. *Exterior protocols* are used to exchange routing information between routing domains. Within your enterprise network, you use an interior routing protocol. It is possible that you will use an exterior routing protocol between your network and your ISP, but even that is unlikely. Most corporate networks are located within the ISP's routing domain, and therefore use an interior routing protocol to talk to the ISP. Following is an overview of three interior protocols—Routing Information Protocol (RIP), Routing Information Protocol Version 2 (RIPv2), and Open Shortest Path First (OSPF)—and one exterior protocol, Border Gateway Protocol (BGP).

Border Gateway Protocol

BGP is the only exterior routing protocol in widespread use. BGP supports *policy–based routing*, which allows you to define organizational or political reasons for choosing a route. The routing metrics used within different routing domains cannot be directly compared. You cannot know exactly how the metrics are determined within another routing domain. You may not even know what interior routing protocols they use. Instead of trusting the technical process used to select the routes, you trust the organization that advertises the routes. You may do this because of the reputation of the organization or because you have a business agreement with them to trust their

routes. This is the basis of policy−based routing.

BGP is a *path−vector protocol*. The path vector provides an end−to−end list of every routing domain along the route, which allows you to decide whether or not you trust the advertisements that come from those domains.

You probably will not run BGP unless you are told to run it by your ISP. Exterior protocols are primarily used externally, and for them to be used successfully, both parties must agree to use the protocol. If your ISP requires you to use BGP, you can be sure your ISP knows how it should be configured. The protocols that you use on your own network are the interior routing protocols.

Routing Information Protocol

RIP is a widely used interior routing protocol. It is included as part of the operating system with many Linux and Unix systems, as well as in the Windows NT/2000 RRAS package. RIP is installed on many systems, and it is easy to configure and run.

RIP defines the "best" route as being the lowest−cost route, which is the one with the lowest routing metric. The *routing metric* is an arbitrary number from 1 to 15 that represents the number of gateways that traffic must pass through to reach the destination. RIP calls each router a "hop" and the metric a "hop count." The best route to a destination is the one that passes through the fewest routers. This technique for determining the best route is called a *distance−vector algorithm*.

When RIP starts, it broadcasts a request for routing information. A router running RIP responds to the request by sending an update packet that contains the destination addresses and associated metrics from its routing table. In addition to responding to requests, routers that run RIP issue update packets every 30 seconds. If a router stops issuing updates for 180 seconds, the other routers on the network assume that it is dead, and delete any routes that go through that router.

RIP processes an update packet in the following manner:

- If the packet contains new routes that are not in the routing table, they are added.
- If the packet contains routes that are lower cost than the same routes in the existing routing table, the old routes are deleted, and the new routes are used. The cost of a new route is determined by adding the cost of reaching the router that sent the update to the cost metric included in the update packet.
- If the packet contains routes that have a cost of 15, those routes are deleted from the routing table if the update came from the gateway used for those routes. For example, if your routing table contains a route to subnet 172.16.50.0 through gateway 172.16.55.36 and it receives a RIP update from 172.16.55.36 with a cost of 15 for the route to 172.16.50.0, your system deletes the route.

RIP has been around for a long time, and it shows its age. Very large networks cannot use RIP because the longest route it allows is 15 hops. This should be large enough for your network, but it is insufficient for large national networks. Additionally, it can take a long time for the routing table to reflect the current state of the network because RIP waits 180 seconds before discarding routes from an inactive router. This can be worsened by the "count to infinity" problem. (Not familiar with "counting to infinity?" See the upcoming sidebar.) More important than either of these problems is the fact that RIP is not equipped to handle network bit masks, which makes it incompatible with current IP address standards.

Counting to Infinity

A problem with the original RIP design is something known as "counting to infinity." Let's use Figure 7.2 to illustrate this problem.

Router B advertises finch as being two hops away because it reaches finch through A. C advertises finch as being three hops away because it reaches it through B. Assume that router A crashes. B no longer gets updates from A, so it doesn't think it can reach finch through A. However, it does see an update from C, saying that finch is three hops away. So B updates its routing table, and now advertises finch as being four hops away. Because C reaches finch through B, it updates its routing table and advertises finch as being five hops away. B then sees the update from C and changes its routing table. This goes on and on until we reach infinity. Luckily, infinity is only 15 for a RIP hop count.

RIP addresses this problem in three ways: *split horizon*, *poison reverse*, and *triggered updates*. The split horizon rule says that a router never advertises a route on the network from which it learned the route. Therefore, C would not advertise that it could reach finch to B because it learned the route from B.

Poison reverse takes this one step further. It says that a router should advertise an infinite distance for a route on the network from which it learned the route. Thus, C would tell B that it cannot reach finch by advertising a hop count of 15 for the route to finch.

With triggered updates, changes to the routing table are immediately advertised instead of waiting the normal 30−second interval between RIP updates. Therefore, even when a count−to−infinity problem does occur, it only takes as long as it takes for the routers to exchange 15 packets. Without triggered updates, counting to infinity can take several minutes.

Split horizon, poison reverse, and triggered updates have effectively eliminated the count−to−infinity problem in RIP. However, they do not address the other problems with RIP. To address those problems, the new routing protocols described next have been developed.

RIP Version 2

RIPv2 enhances the original RIP packet with the addition of a network mask field and a "next hop address" field. The network mask is the bit mask that is used to determine the destination network. Sending the mask with the destination address in the routing update is an essential component of classless IP addresses.

To make your network masks globally available, they must be distributed to other systems. The routing protocol is used to do this, and RIPv2 adds this capability to RIP.

The "next hop address" field provides the address of the gateway. In RIP, only the destination address and the metric are provided. The gateway is always assumed to be the router that sends out the update. The next hop address field specifically provides the address of the gateway, which allows the system that sends the update to be different from the gateway that will handle the route. Thus, RIPv2−capable systems can provide updates for routers that don't run RIPv2. If the next hop address is 0.0.0.0, the router that sends the update is assumed to be the gateway for the route.

In addition to these enhancements that address the biggest problem with RIP, RIPv2 adds a few nice features:

- RIPv2 is completely compatible with RIP because the RIP packet format is unchanged. All of the features of RIPv2 are implemented in unused fields of the original RIP packet. RIP and RIPv2 routers can coexist on a single network without a problem.
- RIPv2 uses multicasting instead of broadcasting to reduce the load on systems that do not want RIPv2 updates.
- RIPv2 provides an authentication scheme that prevents routing updates from a misconfigured host from accidentally being accepted as valid.

Despite its improvements, RIPv2 is still RIP. Therefore, it uses the same distance−vector algorithm for determining the best route, and it limits the diameter of the network to 15 hops. OSPF, a different type of interior protocol, was developed for very large national networks.

Open Shortest Path First Protocol

OSPF is very different from RIP. A router running RIP sends information about the entire network to its neighbors. A router running OSPF floods information about its neighbors to the entire network. *Flooding* means that the router sends the update out of every network port, and every router that receives the update also sends it out of every port except the one it receives it on. Flooding rapidly disseminates routing information to the entire network.

OSPF is called a *link−state protocol* because it creates a graph of the state of all of the links in the network. Every OSPF router creates its graph using the information about all of the routers and their neighbors that flooding distributes throughout the network. Each graph is unique because every router creates the graph with itself as the root of the tree. The graph is built using the Dijkstra Shortest Path First algorithm, hence the name of the protocol. The algorithm builds the graph in this manner:

1. The system starts by installing itself as the root of the graph with a cost of 0.
2. The system installs the neighbors of the system that was just added to the graph. The cost of reaching those neighbors is calculated as the cost of reaching the system just installed plus the cost that system advertises for reaching the neighbors.
3. The system selects the lowest−cost path for each destination. It repeats steps 2 and 3 for every system for which it has information.

Clearly, building a link−state graph for a large network every time a route changes creates a lot of overhead for the router. For this reason, OSPF divides the routing domain up into smaller, more manageable pieces. The entire routing domain is called an *autonomous system*, and the pieces are called *areas*. A special area, called the *backbone area*, is defined to interconnect all of the areas in the autonomous system. Routers within an area only have knowledge of their area and therefore only create a graph of the systems in that area.

OSPF is a much more complex system than RIP, but OSPF is better suited for large networks. However, OSPF is not always needed for an enterprise network, and may not be needed by a small network that uses Linux servers for routers. You may find that RIP is adequate for your needs.

Running RIP with *routed*

Some Linux systems include the routing daemon routed, an implementation of RIP. To run RIP with routed, simply enter **routed.**

routed requires no command−line arguments and no configuration file. The routing daemon listens to the RIP updates on the network to build a functioning routing table. If the computer is a router

(routed assumes it's as if the computer has more than one network interface), the routing daemon broadcasts its own updates. If the computer has only one network interface, routed considers it a host, and the routing daemon does not broadcast routing updates.

Command–line options can be used to change this default behavior, regardless of the number of network interfaces installed in the computer. Use –s to force the daemon to broadcast RIP updates; specify –q to stop it from broadcasting update packets. The –q argument is more useful than the –s argument. Sometimes, you don't want a computer with multiple network interfaces broadcasting routes. However, it is unusual to configure a host with only one network interface to broadcast RIP updates.

routed does not require configuration, but it is possible to use the /etc/gateways file to pass supplemental routing information to the routing daemon.

The */etc/gateways* File

routed broadcasts a RIP request immediately on startup, and uses the information in the RIP updates it receives to build a table. The entire reason for running a routing protocol is to use the information from that protocol to build the routing table. On the surface, adding static routes to a dynamic table doesn't seem to make much sense, and generally there is no reason to do so. But it is possible that there are routers on your network that can't or won't provide RIP updates and that must be added to the table manually. The /etc/gateways file provides that capability, in case you need it.

routed reads the /etc/gateways file during startup, and adds the routes defined there to the routing table. The sample entries from a gateways file are enough to illustrate its purpose because all entries in the file have the same basic format. Listing 7.4 shows two sample entries.

Listing 7.4: A sample */etc/gateways* file

```
$ cat /etc/gateways
net 0.0.0.0 gateway 172.16.55.254 metric 1 active
net 172.16.50.0 gateway 172.16.55.36 metric 1 passive
```

All entries start with the keywords net or host to indicate whether it is a host–specific route or a network route. The keyword is followed by the destination address. (The destination 0.0.0.0 is a special address that stands for the default route.) The destination address is followed by the keyword gateway and the IP address of the external gateway used to reach the destination.

Next come the keyword metric and the cost assigned to this route. Normally, external gateways are given a cost of 1, but this is arbitrary, so you can assign a higher value if you want. Assigning a higher metric, however, makes sense only if you have two routes to the same destination, and if you want to prefer one of those routes over the other.

All entries end with the keywords active or passive. An active router is expected to participate in the exchange of routing updates. If it fails to respond to routing requests and does not periodically broadcast RIP updates, it is removed from the routing table. This is the normal behavior expected of any RIP router.

A passive router does not participate in the exchange of RIP updates. Perhaps the system runs a different routing protocol. Regardless of the reason, it is not required to participate, and is installed in the routing table as a permanent static route.

The first line in our example creates an active default route. This default route is used during the RIP startup period, but after RIP is up and running, this default router is expected to be an active participant in the routing protocol. If you use a default route when running a routing protocol, use an active default route. A static default route can defeat the purpose of a dynamic routing protocol by not allowing the protocol to update the route when network conditions change.

The second line creates a static route to subnet 172.16.50.0 through router 172.16.55.36. Because this is a passive route, 172.16.55.36 does not need to run RIP. The only reason to create such a route would be that 17216.55.36 does not run RIP.

routed is adequate for some small networks. It requires no installation and very little configuration. However, it is antiquated software that is not suitable for many networks. In particular, it does not support classless IP addresses. If you use classless IP addresses, run a modern routing protocol that supports address masks.

Some Linux distributions no longer provide routed, preferring to rely on more modern routing software. Linux offers two different software packages that provide modern routing protocols. The gateway daemon (gated) is a routing software package found on many Unix systems. Zebra is a routing software package from GNU. These packages provide Linux systems with access to interior and exterior routing protocols that normally run only on dedicated routing hardware, such as Cisco routers.

Routing with Zebra

Zebra is a routing software package that provides support for RIP, RIPv2, OSPF, and BGP. In addition, Zebra provides support for IPv6 routing with both the RIPng protocol and the OSPFv6 protocol.

There are internal and external aspects to routing software. On the external side, routing software runs a protocol to exchange routing information with external routers. On the internal side, routing software processes the information learned from the protocol, selects the best routes, and updates the kernel routing table. This internal process increases in complexity when running multiple protocols because it is possible for a router to receive routes to the same destination from different routing protocols. Each protocol uses its own metric for selecting the best route. If each protocol independently updated the routing table, chaos could ensue. To support multiple protocols, and in particular exterior routing protocols, Zebra provides a way to compare incompatible metrics and select the best route. It does this using a value it calls *distance*, which is an arbitrary number from 1 to 255. The higher the distance number of the protocol that provides the route, the less preferred the route. Each protocol has a default distance value assigned to it, which you can override in the configuration.

Zebra uses a modular architecture, with separate programs handling different routing tasks. At this writing, the Zebra suite includes the following key programs:

zebra zebra is the routing manager. It updates the kernel routing table with the routes received from the various routing protocols. All protocols update the table through the zebra routing manager.

ripd ripd provides the RIP and RIPv2 routing protocols.

ospfd ospfd provides the OSPF routing protocol.

bgpd bgpd provides the BGP routing protocol.

vtysh Because each routing protocol and the zebra routing manager are all configured in separate files, configuration can be complex. vtysh is a virtual terminal interface designed for maintaining the router configuration.

ripngd ripngd provides the RIPng routing protocol for use on IPv6 networks.

ospf6d ospf6d provides the OSPFv6 routing protocol for use on IPv6 networks.

This modular architecture is inherently extensible. A new routing protocol can be added by creating a new daemon for the protocol and designing that daemon to interface with the zebra routing manager. Additionally, implementation of the protocol daemon is simplified because zebra handles the internal functions for the new daemon. The new software needs to handle only the external protocol functions.

Installing Zebra

The Zebra suite is available on the Web at http://www.gnu.org/ and via FTP from ftp://ftp.zebra.org/. A beta–testers' website is available at http://www.zebra.org/. At the time of writing, the Zebra software is in beta release. If you prefer to use mature software, see the information on gated later in this chapter. However, for the type of routing applications that should reasonably be undertaken by a Linux system, the beta release of Zebra is more than adequate.

The Zebra software suite is include in some Linux distributions, and can be installed during the initial system installation. Our sample Red Hat Linux system is an example of a distribution that includes Zebra. If it is not already installed, it can be installed on the Red Hat system using the rpm command. Figure 7.3 show a gnorpm query of the Zebra RPM.

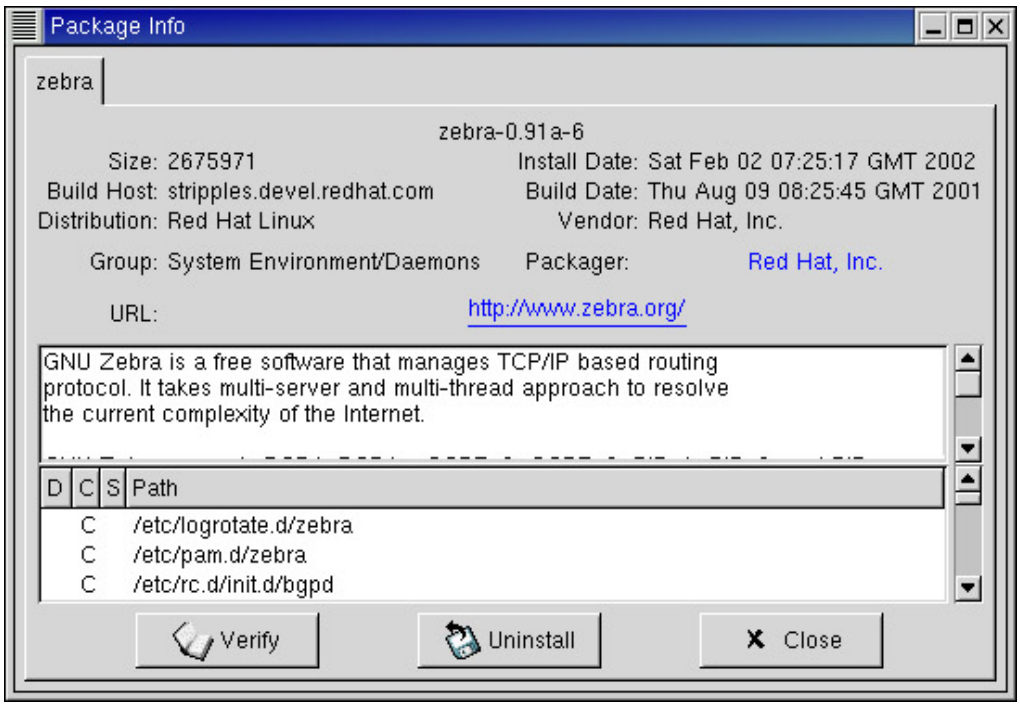


Figure 7.3: Contents of the Zebra RPM

Zebra does not start at boot time, even if installed from the RPM, unless you configure your system to start it. The Zebra RPM includes a separate startup script for the routing manager and each of the routing protocol daemons. Start only those daemons needed to support your routing

configuration. For example, a Red Hat Linux host running only RIPv2 might use the following commands to ensure that routing restarts after a system shutdown:

```
[root]# chkconfig --level 35 zebra on
[root]# chkconfig --level 35 ripd on
```

On the other hand, a Red Hat system acting as an external router might use the following:

```
[root]# chkconfig --level 35 zebra on
[root]# chkconfig --level 35 ospfd on
[root]# chkconfig --level 35 bgpd on
```

If your system does not include startup scripts, the routing manager and the daemons can be started directly, either from the command prompt or from the rc.local script, using the following commands:

```
[root]# zebra -d
[root]# ospfd -d
[root]# bgpd -d
```

The –d option causes the selected program to run as a daemon. Another commonly used option is –f, which defines the path to the program's configuration file.

Of course, our sample Red Hat system does include startup files in the Zebra RPM to ensure that Zebra starts every time the system reboots. Invoke the startup scripts directly to start the routing manager and the routing daemons immediately. For example:

```
[root]# service zebra start
Starting zebra: [  OK  ]
[root]# service ospfd start
Starting ospfd: [  OK  ]
[root]# service bgpd start
Starting bgpd: [  OK  ]
```

These commands show the zebra routing manager, the OSPF daemon, and the BGP daemon starting successfully. This happens only after zebra, ospfd, and bgpd are configured because the Red Hat startup scripts check that the configuration files exist before running the programs. Before you start the daemons, create the required configuration files.

Zebra Configuration Files

Each program in the Zebra suite has its own configuration file. By default, the configuration files are located in the /usr/local/etc directory. On our sample Red Hat system, they are located in the /etc/zebra directory. Each file has a name in the format *program*.conf, where *program* is the name of the program being configured. For example, zebra.conf, ospfd.conf, and ripd.conf are all standard configuration filenames. Of course, you can name the configuration file anything you want and place it anywhere on the system because the file can be identified to the program using the –f command–line option when the program is run. However, it is best to stick with standard names, so that others can easily find the files when performing maintenance.

Because the routing manager and the various protocols are configured separately, and because modern routing protocols can be very complex, there are hundreds of configuration commands available for Zebra. The "Command Index" section of the HTML documentation that comes with Zebra lists them all. We do not repeat that list here, for a couple of reasons. First, the laundry list of commands is already provided in the online Zebra documentation—Zebra provides excellent HTML

and info documentation files, and quick reference man pages. Second, most of these commands never appear in a Linux router configuration. The best way to learn about Zebra configuration is to look at some reasonable examples.

The *zebra.conf* File

The zebra routing manager is required if you want to use any of the Zebra routing daemons. zebra maintains the kernel routing table, maintains the network interface list, defines the static routes, and manages the sharing of information between the different routing protocols.

zebra is configured by the zebra.conf file. Listing 7.5 shows a sample zebra.conf file for a Linux system using two Ethernet interfaces:

Listing 7.5: Sample *zebra.conf* File

```
! The hostname of this router
hostname subnet60gw
! The password required for vtysh access
password Wats?Watt?
! The password required for privileged vtysh commands
enable password CHLLlns
! The first network interface
interface eth0
    ip address 172.16.60.1/24
    multicast
! The second network interface
interface eth1
    ip address 172.16.1.9/24
! The path to the log file
log file /var/log/zebra.log
! A sample static route
ip route 172.16.50.0/24 172.16.1.50
!
```

Comments in the various Zebra configuration files begin with an exclamation mark (!). The comments in Listing 7.5 explain the commands that follow them.

The sample file in Listing 7.5 is larger and more complex than most. It was made that way to provide a full range of examples, but it is worthwhile to remember that your zebra.conf file will probably be smaller.

The hostname command defines the hostname of the router. This is the name that will be used in routing protocol exchanges that use router names. zebra.conf files often start with this command.

Next come two commands that define passwords. The password command is necessary if you plan to use the vtysh interface. If this password is not defined, the vtysh interface is disabled for zebra. The second password provides greater configuration control to the vtysh interface. Without the password defined by the enable password command, the vtysh interface can only be used to query zebra for information. With this second password, the zebra configuration can be controlled from the interface. We look at vtysh in more detail later.

The first interface command defines the configuration of eth0. The command contains two clauses: ip address, which defines the address of the interface; and multicast, which turns on the multicasting flag for the interface. The multicast flag might be used if you planned to use a

multicasting protocol, such as RIPv2, on this interface. The second interface command is much like the first, except that it does not set the multicast flag.

The log file command defines the file into which zebra should write its logging information. In Listing 7.5, the log is written to /var/log/zebra.log. Alternatives to logging to a specific file are to use the log stdout command to log to standard output or the log syslog command to log through syslogd.

The last command in Listing 7.5 defines a static route. Although static routes may be distributed by protocols configured in other files, such as ripd.conf, the static routes are defined in the zebra.conf file. The keywords ip route are followed by the destination of the route and then the router used to reach that destination. In Listing 7.5, the destination is network number 172.16.50.0. A 24–bit address mask is being used to match addresses to the destination. The gateway for the route is 172.16.1.50. Of course, this is just an example. You probably won't use static routes when you run routing protocols.

Two lines in the zebra.conf file shown in Listing 7.5 define passwords for the vtysh interface. This file should therefore be readable only by administrators. Before going on to the configuration of individual routing protocols, let's look at how these passwords are put to use for the vtysh interface.

Using *vtysh*

The vtysh tool provides an interactive interface into the zebra routing manager and each routing daemon. vtysh allows you to examine and modify the configuration of each program in the Zebra suite. Listing 7.6 shows a vtysh session in which the current zebra configuration is examined. In a later example, this configuration will be modified with vtysh.

Listing 7.6: Examining *zebra.conf* through the *vtysh* Interface

```
[root]# cat /etc/zebra/zebra.conf
password Wats?Watt?
enable password CHLLlns
[root]# service zebra start
Starting zebra: [  OK  ]
[root]# telnet localhost zebra
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

Hello, this is zebra (version 0.91a).
Copyright 1996-2001 Kunihiro Ishiguro.

User Access Verification
Password: Wats?Watt?
junko.foobirds.org> enable
Password: CHLLlns
junko.foobirds.org# write terminal

Current configuration:
!
password Wats?Watt?
enable password CHLLlns
!
interface lo
!
interface eth0
!
line vty
```

```
!
end
```

Listing 7.6 starts our sample vtysh session. We begin by displaying the current zebra.conf file. A minimal zebra.conf file must exist before zebra can be configured through the vtysh interface. The minimal configuration file must contain the passwords required for vtysh access. The first password is required to start the vtysh session. The second password is required to enter *enable mode*. You must be in enable mode to view or modify the configuration.

In addition to requiring a minimal configuration, the daemon must be up and running before it can be configured from the vtysh interface, as illustrated by the service zebra start command shown in Listing 7.6. The vtysh interface is invoked by making a telnet connection to the running daemon. Notice in Listing 7.6, we telnet to the zebra port on the local host. For this to work, the port number must be defined in the /etc/services file. If the zebra port is not defined on your system, you must telnet to port 2601. The port numbers from the services file on our sample system are shown in Listing 7.7.

Listing 7.7: The Port Numbers Used by the Zebra Suite

```
$ tail -8 /etc/services
# Local services
zebrasrv        2600/tcp                    # zebra service
zebra           2601/tcp                    # zebra vty
ripd            2602/tcp                    # RIPd vty
ripngd          2603/tcp                    # RIPngd vty
ospfd           2604/tcp                    # OSPFd vty
bgpd            2605/tcp                    # BGPd vty
ospf6d          2606/tcp                    # OSPF6d vty
```

Every routing daemon has its own port number. Before any of these can be configured by vtysh, a minimal configuration file containing the necessary passwords must be created for the daemon, and the daemon must be up and running. For example, to use the vtysh interface with ripd, you must first create a minimal ripd.conf file and start ripd. The first two steps shown in Listing 7.6 apply to all of the daemons.

After connecting to the zebra routing manager in Listing 7.6, the first password is entered in response to the Password: prompt. This password starts the vtysh session. The enable command is entered to invoke enable mode. We again receive a password prompt, but this time it is for the password defined by the enable password command in the zebra.conf configuration file. After that password is entered, enable mode commands can be used.

The first enable mode command in Listing 7.6 is write terminal, which displays the current configuration on the terminal screen. Notice that this configuration is different from the two–line configuration shown in response to the cat command. The write terminal command displays the configuration that is being used, which includes various defaults. Notice that all of the interfaces on the system, lo and eth0, are used by default. Also note the line vty command. This command appears in the configuration file when the configuration can be modified from the vtysh interface, which is the default.

Having examined the existing configuration, we are ready to customize it to our needs. Listing 7.8 is a continuation of the session shown in Listing 7.6.

Listing 7.8: Reconfiguring *zebra.conf* through the *vtysh* Interface

```
junko.foobirds.org# configure terminal
junko.foobirds.org(config)# hostname junko
junko(config)# log file /var/log/zebra.log
junko(config)# interface eth0
junko(config-if)# ip address 172.16.20.3/24
junko(config-if)# multicast
junko(config-if)# exit
junko(config)# exit
junko# write file
Configuration saved to /etc/zebra/zebra.conf
junko# exit
Connection closed by foreign host.
[root]# cat /etc/zebra/zebra.conf
!
! Zebra configuration saved from vty
!   2002/05/28 15:22:17
!
hostname junko
password Wats?Watt?
enable password CHLLlns
log file /var/log/zebra.log
!
interface lo
!
interface eth0
 ip address 172.16.20.3/24
 multicast
!
line vty
!
```

Listing 7.8 picks up where Listing 7.6 ended. We are still in enable mode, and all of the vtysh commands in Listing 7.8, except for the exit commands, can be entered only in enable mode.

The configure terminal command tells the system that configuration commands will be entered from the vtysh interface. As each configuration command is entered, vtysh checks it for syntax errors, which is one of the best reasons for entering configuration commands through the vtysh interface. The commands hostname, interface, ip address, and multicast (shown in Listing 7.8) are basic configuration commands found in the zebra.conf file. In fact, all of these commands were discussed earlier when we looked at the zebra.conf example in Listing 7.5.

Notice how the vtysh command prompt changes to indicate the different modes of operation. Listing 7.6 shows that the prompt for standard mode, called *view mode*, is a dollar sign ($), and that the prompt for enable mode is a hash mark (#). When the session enters configuration mode in Listing 7.8, the prompt indicates this with a (config) string. Furthermore, the prompt string changes to indicate what is being configured. For example, the string (config–if)indicates that an interface is being configured. Also notice that each time an exit command is entered, vtysh exits the current mode until the final exit command ends the session.

The new configuration is stored in the zebra.conf file by the write file command. The final exit command in Listing 7.8 ends the session that began in Listing 7.6. A cat of the zebra.conf file shows that it has changed substantially from the zebra.conf file that was displayed at the start of Listing 7.6. The original zebra.conf file is not gone, however. vtysh saves the previous configuration file with a .sav extension when it writes the new file. The old zebra.conf file is now stored in /etc/zebra

with the name zebra.conf.sav.

The vtysh interface is a tool intended to simplify a complex configuration problem. In general, however, Linux routers do not have highly complex configurations; and the configurations, once set, do not change frequently. Additionally, a small configuration file must be created before vtysh can be used, and the final configuration files are often not much larger than the minimal ones that must be created by hand. For these reasons, you may find vtysh more useful as a tool for examining the router configuration than as a tool for modifying the configuration. Personally, I like vtysh; but if you prefer, you can build Zebra configuration files with your favorite text editor. It's up to you.

The following sections show a few reasonable Linux Zebra configurations. We configure a host to run RIPv2, an interior router to run RIPv2 and OSPF, and an exterior router to run OSPF and BGP.

Running *ripd*

Routing protocols are not limited to routers. It is possible to need a routing protocol on a Linux host. Suppose that you have a host on a network in which routing updates are distributed via RIPv2. This system is not a router, but because it is on a network segment with more than one router, you decide to configure it to listen to the RIPv2 updates that the routers are broadcasting. Listing 7.9 shows a possible ripd.conf file for this host.

Listing 7.9: A Sample *ripd.conf* File

```
! Enable RIPV2, but don't send updates
! Check that packets are authentic
interface eth0
  ip rip authentication string EZdozIt
!
router rip
  passive-interface eth0
!
```

The RIPv2 configuration is very simple. The command router rip enables RIP. By default, Zebra uses RIPv2, which is capable of handling address masks and is compatible with RIP version 1. To force Zebra to use RIP version 1, a version 1 clause could be used with the router rip command, but using RIP version 1 is generally ill advised.

The passive−interface clause is used because this host listens to routing updates, but does not send routing updates. This is equivalent to the −q option mentioned earlier for the routed command. passive−interface is used on hosts that listen for updated; routers that actively participate in the routing exchange use the network clause. The network clause uses the interface name to identify the interface over which routing updates are exchanged; for example, network eth0. Alternatively, it can use an IP address to identify the systems with which routing updates are exchanged. The IP address is defined with an address mask. For example, network 172.16.60.0/24 would exchange routing updates with any system on subnet 172.16.60.0. We will see the network clause in action when we configure an interior router.

The interface statement is used to configure the network interface for RIP. In Listing 7.9, a RIPv2 authentication mode is set for the interface. In this example, a simple clear text password is used. Clear text passwords are used to help the router avoid accepting updates from misconfigured systems; it is not a security method. Stronger update authentication is available in the form of MD5 authentication.

In addition to the ripd.conf file shown in Listing 7.9, the host needs a zebra.conf file. The zebra.conf file we created for our sample host is shown in Listing 7.10.

Listing 7.10: A *zebra.conf* File for a Linux Host

```
hostname grebe
password lOOK!c?
log file /var/log/zebra.log
!
interface eth0
    ip address 172.16.60.2/24
    multicast
!
```

This file is simpler than the zebra.conf file shown in Listing 7.5. It defines the hostname used by this host and the path to the log file, but it defines only one vtysh password. This password allows interactive queries of the routing configuration, but it does not allow the configuration to be changed. The interface command in Listing 7.10 defines the interface over which this host listens to RIPv2 updates.

The RIPv2 updates that this host uses to build its routing configuration come from the routers on the subnet. In the next section, we look at the sample configuration of such a router.

Running *ospfd*

In this section, a router is configured to send RIPv2 packets on one subnet—the one that it shares with the sample host configured in Listing 7.9—and OSPF link–state advertisements on another subnet that it shares with other routers. For this configuration, we create a zebra.conf file to configure the routing manager, a ripd.conf file to configure the RIPv2 protocol daemon, and an ospfd.conf file to configure the OSPF protocol daemon. The zebra.conf file is almost identical to the one shown in Listing 7.5.

Listing 7.11: A *zebra.conf* File for a RIP/OSPF Router

```
hostname subnet60gw
password Wats?Watt?
enable password CHLLlns
log file /var/log/zebra.log
!
interface eth0
    ip address 172.16.60.1/24
    multicast
!
interface eth1
    ip address 172.16.1.9/24
!
```

The zebra.conf file in Listing 7.11 contains everything that was found in Listing 7.5, except for the static route. All of this was explained earlier, so there is no need to explain it again here.

The ripd.conf file used for this configuration is shown in Listing 7.12.

Listing 7.12: A *ripd.conf* File for a RIP/OSPF Router

```
! Enable RIPV2
! Advertise routes learned from OSPF with a cost of 5
! Use simple authentication for updates
password RIPItup
enable password RaceitUP
!
interface eth0
   ip rip authentication string EZdozIt
!
router rip
   redistribute ospf metric 5
   network eth0
!
```

This configuration is very similar to the ripd.conf file shown in Listing 7.9, but there are differences. First, we have placed the passwords in this file so that the RIP router configuration can be maintained through the vtysh interface.

Second, because this is a router, it sends routing updates. Therefore, we use a network clause under the router rip statement to specify the network on which routing updates will be distributed. As described earlier, the network can be defined by an interface name or by an IP address. In Listing 7.12, it is defined by an interface name.

The third difference is that this configuration contains a redistribute clause, which defines whether routes learned from OSPF will be advertised to RIP neighbors and what RIP cost will be assigned to those routes. Routes learned from OSPF do not have a valid RIP cost. The metric defined on the redistribute clause is used as a default metric when OSPF routes are advertised to RIP neighbors. The redistribute clause in Listing 7.12 tells RIP to advertise routes learned from OSPF with a cost of 5. Of course, for RIP to learn routes from OSPF, ospfd must be properly configured.

Listing 7.13 shows a possible ospfd.conf configuration for this router.

Listing 7.13: A Sample *ospfd.conf* File

```
! Enable OSPF; connect to the backbone area
! Use simple authentication
password Mutt-N-Jeff
enable password SURtest
!
interface eth1
 ip ospf authentication-key UTrustME
 ip ospf priority 5
!
router ospf
 ospf router-id 172.16.1.9
 network 172.16.1.0/24 area 0
!
```

The interface statement sets two OSPF parameters. The ip ospf authentication–key clause defines the clear text string used to identify valid OSPF advertisements. Like the clear text string used for RIPv2 authentication, this string is intended to prevent accidental updates from misconfigured systems; it is not intended to provide real security. OSPF supports MD5 for stronger authentication.

The ip ospf priority clause defines the priority number this system uses when the area elects a designated router. In Listing 7.13, the priority number is set to 5—possible values are 0 to 255. The

larger the priority number, the less likely the router will be elected the designated router. Give your most powerful router the lowest priority number.

A designated router is used to reduce the size of the link–state database and thus the complexity of calculating the Dijkstra graph of the area. The designated router treats all other routers in the area as neighbors, but all other routers treat only the designated router as a neighbor.

To understand how this reduces the size of the link–state database, think of a network of five routers. Without a designated router, all five routers advertise four neighbors, for a total of 20 neighbors in the database. With a designated router, only that router advertises four neighbors. The other four routers advertise one neighbor for a total of eight neighbors in the database. The larger the network, the more important it is to use a designated router.

The router ospf statement enables the OSPF protocol. In Listing 7.13, the router ospf statement contains two clauses. The ospf router–id clause defines the router identifier used for OSPF advertisements. Normally, this is set to the primary address of one of the interfaces used for OSPF. On this sample router, only one network interface is being used for OSPF, so the address of that interface is used as the router identifier.

The network clause identifies the network over which OSPF routes will be exchanged. In the ospfd.conf file, the network on the network clause is always defined by an IP–address/network–mask pair. The network clause also identifies the area of which the network is part. Remember that OSPF divides the autonomous system into areas, and every OSPF router must connect to some area. As mentioned earlier, the area that interconnects all other areas within the routing domain is called the backbone area. The number assigned to the backbone area is 0. Therefore, the network clause in Listing 7.13 specifies that this router is connected to the backbone area.

A simple OSPF configuration, such as the one shown in Listing 7.13, should be adequate for any Linux system that needs to run OSPF. Much of the configuration information will come from the network designer who defines your routing hierarchy. The area you connect to, the type of authentication used, the authentication password, and the priority number of your system are all design decisions that will be made before your network even begins to run OSPF.

As a final example of configuring the Zebra suite, we configure the router that attaches the OSPF network to the outside world via BGP.

Running *bgpd*

The zebra.conf file and the ospfd.conf file used on the BGP router are almost identical to the files shown in Listings 7.11 and 7.13, as the following cat commands show:

```
[root]# cat /etc/zebra/zebra.conf
hostname externalgw
password BILL&ted
enable password 4138doc
log file /var/log/zebra.log
!
interface eth0
    ip address 172.16.1.5/24
    multicast
!
interface eth1
    ip address 26.10.105.4/8
!
```

```
[root]# cat /etc/zebra/ospfd.conf
! Enable OSPF; connect to the backbone area
! Use simple authentication
password a-DA-zip
enable password TX4123
!
interface eth0
   ip ospf authentication-key UTrustME
   ip ospf priority 10
!
router ospf
   ospf router-id 172.16.1.5
   redistribute bgp
   network 172.16.1.0/24 area 0
!
```

Passwords and addresses have changed. Beyond these cosmetic changes, there is nothing new in the zebra.conf file that needs to be discussed.

The ospfd.conf file has two interesting changes. First, this router is assigned a higher priority number. (Given the work it will be doing to handle BGP, we decided not add the burden of being the designated router for OSPF.) Second a redistribute bgp clause has been added because we want the router to advertise the routes it learns from BGP to the other OSPF routers.

The largest new part of this configuration is the bgpd.conf file. In Listing 7.14, a possible bgpd.conf file for a router to attach the OSPF backbone area (described for Listing 7.13) to an external autonomous routing domain using BGP is defined.

Listing 7.14: A Sample *bgpd.conf* File

```
! Enables BGP
! Our ASN is 264; our neighbors are in ASN 164
! Advertise directly connected routes and routes
! learned from OSPF
password BDRwar
enable password awtoMA
!
router bgp 249
 redistribute connected
 redistribute ospf
 neighbor 26.6.0.103 remote-as 164
 neighbor 26.20.0.72 remote-as 164
!
```

The router bgp 249 statement enables BGP and assigns the autonomous system number (ASN). BGP exchanges routing information between autonomous systems; thus, an ASN is required for BGP to function. Listing 7.14 says that the ASN of our autonomous system is 249.

Note To exchange routing data between official routing domains, you need an official ASN; you can't just make one up, and you *can't* use the ones in this example—they are the official ASNs of two government networks I worked on. For information on filing the paperwork to apply for your own official ASN, go to http://www.iana.org/, and follow the links to the appropriate registry for your part of the world. If you use BGP to link together independent networks within a single autonomous system, and the routing data stay within that autonomous system, use one of the ASN numbers reserved for private use. The numbers

reserved for private use are 64512 to 65534.

The two redistribute clauses define the routes that will be advertised to our BGP neighbors. redistribute connected tells the router to advertise routes for all networks to which the router is directly connected. redistribute ospf tells the router to advertise routes that it learns from OSPF.

The neighbor clauses define the two BGP neighbors that this router should peer with. The neighbors are defined by an IP address, and the external autonomous system to which they belong is identified by the remote–as parameter. In Listing 7.14, the external ASN is 164. Do *not* use this number or 249. They are both officially assigned to government networks.

Zebra is beta software that has only recently begun shipping with Linux distributions, and it is not the only choice for routing software. Many systems still use gated, which is our next topic.

Using *gated*

Despite the fact that Red Hat Linux uses Zebra as its default routing software, many other Linux distributions ship with gated. If you don't have gated software with your distribution, a commercial version can be obtained from the Internet at http://www.gated.org/. Also, at this writing, you can still find and download a precompiled Linux gated binary from an online repository. However, if your distribution doesn't include gated, this is a good time to transition to Zebra. If you do have gated, and you want to use it, read on.

Like Zebra, gated supports many of the most advanced routing protocols. Unlike Zebra, the free version of gated combines these protocols in a single large program. gated was created to allow a system to run multiple routing protocols and to combine the routes learned from those protocols. It does this using a *preference value.* A gated preference value is an arbitrary number between 0 and 255 that indicates whether one source of routing information is preferred over another. The sources of information can be different routing protocols, different interfaces, different routers, and different routing domains. The lower the preference number, the more preferred the source. The default preferences used for routing protocols are shown in Table 7.1.

Table 7.1: Default *gated* Preference Values

Route Source	Preference Value
Direct route	0
OSPF	10
IS–IS Level 1	15
IS–IS Level 2	18
Internally generated default	20
ICMP redirect	30
Routes learn from the route socket	40
Static route	60
SLSP routes	70
HELLO routes	90
RIP	100
Point–to–Point interface routes	110
Routes through a downed interface	120
Aggregate and generate routes	130

OSPF ASE	150
BGP	170
EGP	200

Given the preference values in Table 7.1, a route through a network interface to a directly connected network is the most–preferred route; and a route learned from EGP, an obsolete exterior routing protocol, is the least–preferred route. OSPF is listed in the table twice. OSPF ASE routes were learned by OSPF from an external autonomous system. Because the ASE routes come from another routing domain, the metrics in those routes do not receive the same level of trust as the metrics in interior routes. In fact, the three lowest–preference routing sources—EGP, BGP, and OSPF ASE—all get the routes from external routing domains.

You can modify these default preferences when you configure gated, but you probably won't need to. The defaults work well for most configurations. In part, this is because a general–purpose system such as Linux isn't used for extremely complicated and demanding routing situations. Instead, dedicated router hardware is used. For less–demanding applications, such as providing the gateway to a single subnet, Linux is an excellent choice.

Installing *gated*

The gated software is part of some Linux distributions, and when it is, gated is often installed during the initial system installation. If the gated package was not installed during the initial installation, install it now. In this section, we use Red Hat Linux 7.1 as our sample system because the examples in this book are Red Hat–based, and 7.1 was the last release of Red Hat that shipped gated as its default routing software.

On a Red Hat 7.1 system, use RPM to install the software from the CD–ROM. Figure 7.4 shows the result of a gnorpm query after gated is installed.

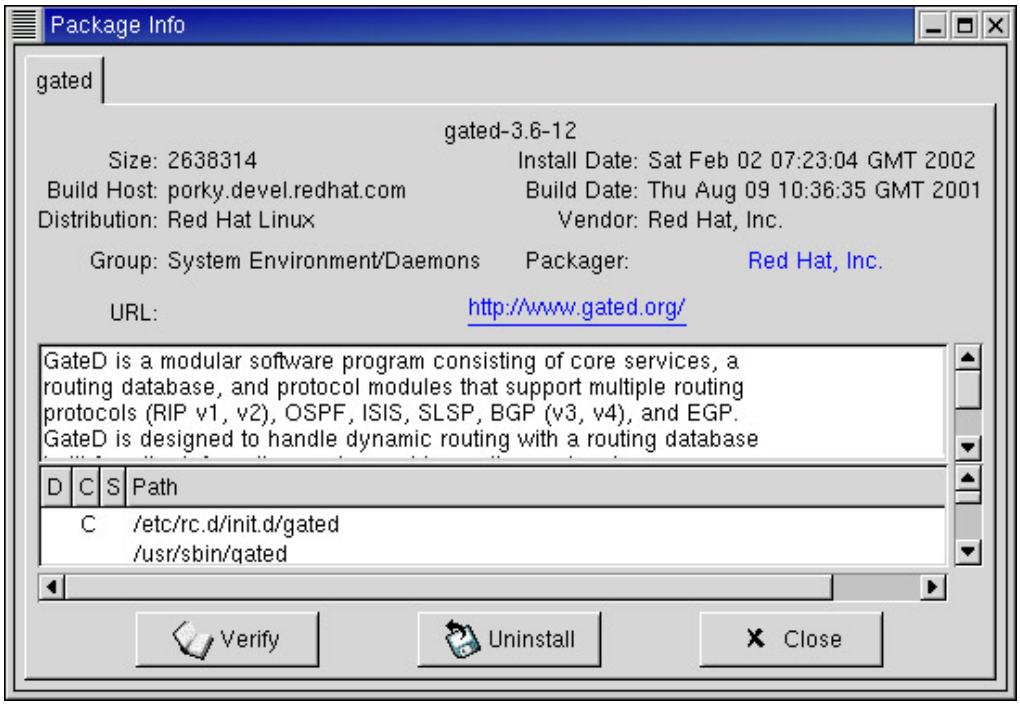


Figure 7.4: Installing gated with gnorpm

After gated is installed, use a tool such as tksysv or chkconfig to enable it. The following example shows gated being enabled for run levels 3 and 5:

```
[root]# chkconfig --list gated
gated   0:off  1:off  2:off  3:off  4:off  5:off  6:off
[root]# chkconfig --level 35 gated on
[root]# chkconfig --list gated
gated   0:off  1:off  2:off  3:on   4:off  5:on   6:off
```

Of course, you don't have to wait for the system to reboot to run gated. On a Red Hat 7.1 system gated is started by the script /etc/init.d/gated, which can be run from the shell prompt, as follows:

```
[root]# service gated start
Not starting gated:  [  OK  ]
```

Not starting! This is not what we expected. When the script runs successfully, the message displayed is "starting gated". However, the script file will not attempt to start gated unless a gated configuration is provided. gated is configured through the /etc/gated.conf file. If we had scrolled through the list of files that Figure 7.4 shows for the gated RPM, we would have found that Red Hat does not provide a preconfigured gated.conf file. If you want to run gated, you must build your own configuration.

The *gated.conf* File

At startup, gated reads the gated.conf file. The file contains configuration statements that tell gated which routing protocols should be run and how they should be configured. There are several types of configuration statements:

- Options statements
- Interface statements
- Definition statements
- Protocol statements
- Static statements
- Control statements
- Aggregate statements

Not all of these statements are required for a configuration, but when they are used, the statements must appear in the order listed here.

These statements can be divided into two groups: statements you probably won't use and statements you might use. Among the statements you're unlikely to use are the options statements, the static statements, the control statements, and the aggregate statements:

- The options statements set parameters such as nosend (don't send any routing information) and noresolv (don't use DNS), which are used only for special configurations.
- The static statements define the static routes that should be included in the routing table. Generally, when you run a routing protocol, you don't need to define static routes.
- The control statements are used to define the routing policy. They are primarily used when passing routing between routing domains. Although it is unlikely that you will be using Linux for this, an example of using control statements appears later because it is one of the key features of gated.
- The aggregate statements are used to aggregate routes within regional and national networks to reduce the number of routes exchanged between national networks.

The statements that you are more likely to use in a Linux gated configuration are defin−ition statements, interface statements, and protocol statements:

- The definition statements define invalid destination addresses, the autonomous system number for exterior routing protocols, and the router IP address for BGP and OSPF. You'll see some examples of definition statements later.
- The interface statements are used to define the characteristics of your router's network interfaces. You will see this statement in our sample configuration.
- The protocol statements are the heart of the gated.conf file. Every routing protocol available in gated has a protocol statement. Use the protocol statement to configure the routing protocol for your network.

gated is a complex system that can handle many different routing configurations. The configuration language has a multitude of options. Details of the language are covered in the online manual at http://www.gated.org/—and in printed form in *TCP/IP Network Administration*, by Craig Hunt (O'Reilly, 2002).

Routers running on Linux systems, however, usually don't require all of these configuration options. The best way to understand the gated configuration commands is to look at a few reasonable Linux configurations.

Running RIPv2 with *gated*

gated can be used to configure a host to listen to RIPv2 router updates. This configuration performs the same function as the ripd configuration shown in Listing 7.9. Listing 7.15 is a possible gated configuration for this situation.

Listing 7.15: A *gated* RIPv2 Configuration

```
#  enable rip, don't broadcast updates,
#  listen for RIP-2 updates on the multicast address,
#  check that the updates are authentic.
#
rip yes {
        nobroadcast ;
        interface 172.16.60.2
             version 2
             multicast
             authentication simple "EZdozIt" ;

} ;
```

The comments at the beginning of the configuration file help to explain the configuration. Other than these comments, the entire file is one protocol statement. All of the lines enclosed inside the curly braces ({}) are part of the RIP protocol statement.

The statement begins with rip yes, which enables the RIP protocol. The nobroadcast clause tells the system not to send RIP update packets; it will just listen to the packets provided by the routers. If your system is a router instead of a host, delete this clause and it will send updates.

The interface clause defines the interface the routing protocol should use and the characteristics of the interface. In this case, the interface is identified by its IP address. Hosts have only one interface. If this were a router that ran RIPv2 on all interfaces, you could provide a comma–separated list of all interface IP addresses or the keyword all to indicate that all interfaces should be used.

The interface clause also contains some parameters that are specific to RIPv2. The parameter

version 2 explicitly tells gated to run RIPv2. The keyword multicast says to listen for updates on the RIPv2 multicast address. Finally, the authentication parameter defines the type of RIPv2 router authentication that will be used. In this case, we use simple password authentication. The password is EZdozIt.

This example provides a RIPv2 host configuration that could be used on any system with only slight modifications. Our sample host receives its RIP updates from local routers. In the next section, one of these routers is configured.

Running OSPF with *gated*

Listings 7.13 and 7.14 define the configuration of a router that uses RIPv2 on one subnet and OSPF on another. That same configuration can be replicated with gated. Listing 7.16 is a sample gated OSPF router configuration.

Listing 7.16: A *gated* OSPF/RIPv2 Interior Router Configuration

```
#  Don't time-out subnet 60
interfaces {
    interface 172.16.60.1 passive ;
} ;
# Define the OSPF router id
routerid 172.16.1.9 ;
# Enable RIP-2; announce OSPF routes to
# subnet 60 with a cost of 5.
rip yes {
    broadcast ;
    defaultmetric 5 ;
    interface 172.16.60.1
         version 2
         multicast
         authentication simple "EZDozIt" ;
} ;
# Enable OSPF; subnet 1 is the backbone area;
# use password authentication.
ospf yes {
     backbone {
         authtype simple ;
         interface 172.16.1.9 {
              priority 5 ;
              authkey "UTrustME" ;
              } ;
          } ;
} ;
```

The configuration begins with an interfaces statement. It tells the router that the systems on subnet 60 may not provide RIPv2 updates. Normally, if no routing information is received on an interface, the interface is marked as inactive and assumed to be "down." This statement ensures that the interface is not assumed to be "down" just because the hosts on the subnet do not advertise RIPv2 updates. Unlike the interface clause in Listing 7.15, this statement is not subordinate to a protocol statement.

The routerid definition statement defines the address that will be used to identify this router for OSPF. Routers have more than one network interface and therefore more than one IP address. To ensure that the correct address is used in the OSPF link–state advertisements, specifically define the OSPF routerid.

The next statement is a protocol statement that enables RIPv2. Except for two differences, it is identical to the statement used in Listing 7.15 to enable RIPv2 for the host. The first difference is that the router will advertise RIPv2 routes, as indicated by the broadcast keyword. The second difference is that the configuration defines the RIP metric used to advertise routes learned from other protocols, which can be any valid RIP metric value. This router learns routes from OSPF that do not have a valid RIP metric. The defaultmetric clause tells gated to use a cost of 5 to advertise those routes in RIP updates. This clause is required to make the routes learned from OSPF available to the RIPv2 system. Without it, the OSPF routes are considered "unreachable" by the RIPv2 systems.

The final protocol statement in the sample gated.conf file shown in Listing 7.16 enables OSPF. This router connects to the backbone area, as indicated by the keyword backbone. If the router was not connected to the backbone, the area it was connected to would be defined here (for example: area 1). The number that identifies the area is the number that you define when you design the area hierarchy of your OSPF routing domain.

The OSPF protocol is also using simple password authentication, as indicated by the authtype simple clause. The interface clause identifies the interface over which OSPF runs and the protocol characteristics related to that interface. The authkey "UTrustME" clause defines the password used to authenticate OSPF routers in this area.

The priority 5 clause defines the priority number this system uses when the area elects a designated router. The purpose of the priority number was described in the discussion of Listing 7.13 as part of the ospfd configuration.

As a final example of running gated, let's configure a Linux system to run an exterior gateway protocol.

Running BGP with *gated*

In this section, a router is configured to connect the OSPF backbone area described in the preceding section to an external autonomous system using BGP. The configuration for this router is shown in Listing 7.17.

Listing 7.17: A *gated* OSPF/BGP Exterior Router Configuration

```
# Defines our AS number for BGP
autonomoussystem 249;

# Defines the OSPF router id
routerid 172.16.1.5;

# Disable RIP
rip no;

# Enable BGP
bgp yes {
   group type external peeras 164 {
        peer 26.6.0.103 ;
        peer 26.20.0.72 ;
        };
};

# Enable OSPF; subnet 1 is the backbone area;
# use password authentication.
ospf yes {
```

```
     backbone {
         authtype simple ;
         interface 172.16.1.5 {
              priority 10 ;
              authkey "UTrustME" ;
              } ;
          } ;
};

# Announce routes learned from OSPF and route
# to directly connected network via BGP tp AS 164
export proto bgp as 164 {
     proto direct ;
     proto ospf ;
};

# Announce routes learned via BGP from
# AS number 164 to our OSPF area.
export proto ospfase type 2  {
     proto bgp as 164  {
          all ;
          };
};
```

The definition statements at the beginning of Listing 7.17 define the autonomous system number (ASN) for BGP and the router identifier for OSPF. The autonomoussystem statement says that the ASN of our autonomous system is 249. (As noted earlier, do not use ASN 249 in your configuration.)

The first protocol statement in the sample configuration disables RIP. This router does not run RIP. On one side, it connects to the OSPF backbone area; on the other side, it connects to an external routing domain with BGP.

The second protocol statement enables OSPF. There is nothing new here; it looks almost identical to the OSPF protocol statement you have already seen in Listing 7.16.

The next protocol statement enables BGP. The group clause defines the characteristics of a group of BGP neighbors, which gated calls *peers*. The IP address of each peer is identified within the group clause by a peer clause.

One of the characteristics defined by the group clause is the type of BGP session to establish with the peers. In the example, BGP is used as a classic exterior gateway protocol, thus the type external parameter. BGP can be used for other purposes. As noted in the section on bgpd during the discussion of ASN numbers, it is possible to use BGP to distribute routes within a routing domain instead of between routing domains. When it is used in this way, BGP is referred to as *internal BGP (IBGP).* Here is the group clause from the sample file in Listing 7.17:

```
group type external peeras 164
```

This says that BGP will run as an exterior routing protocol and that the ASN of the external autonomous system with which it will communicate is 164. Of course, this should be the real ASN number of your BGP neighbor.

On the other hand, assume that you have a large, far–flung enterprise internet. Within that enterprise network are several networks that run OSPF as an interior routing protocol, and have

their own independent connections to the global Internet. You could use internal BGP to move routing information between the individual networks that make up your enterprise network. An example of the group statement for such a configuration is

```
group type igp peeras 64550 proto ospf
```

This says that BGP will run as an internal gateway protocol, the ASN 64550 will be used within your enterprise network, and the routers you are exchanging updates with learn their routes through OSPF.

The sample configuration concludes with two control statements: the export statements that define the routing policy. The first statement defines which internal routes are advertised to the external world. It tells gated to export to the autonomous system identified by ASN 164 all direct routes and all routes that the local router learns from OSPF.

The final export statement defines the routes that gated accepts from the external world and advertises on the internal network. The first line of this statement is

```
export proto ospfase type 2
```

This tells gated to advertise the routes via the OSPF protocol as autonomous system external (ASE) routes, which means the routes are clearly marked as routes learned from an external source. The type 2 parameter indicates that the routes come from a protocol that does not use a metric that is directly comparable to the OSPF metric. The alternative is type 1, which means that the metrics are directly comparable. However, BGP is a path−vector protocol, not a link−state protocol, and its metrics are not directly comparable to those used by OSPF. You know the routes were learned from BGP by looking at the rest of the export statement:

```
proto bgp as 164 {
   all ;
   }
```

This says that the routes being exported were received via BGP and that they come from the autonomous routing domain identified by ASN 164. Furthermore, the keyword all in this clause says that gated should accept all routes from that autonomous system. Instead of the keyword all, you can use specific addresses to accept only specific routes or the keyword restrict to block all routes.

Note These discussions of OSPF and BGP show that routing can be a very complex topic. If you need to use a routing protocol that is more complicated than RIPv2, read more about it, and design your routing architecture before you try to configure a system. See *Internet Routing Architectures* by Bassam Halabi (Cisco, 1999) and *IP Routing Fundamentals* by Mark Sportack (Cisco, 1999) for additional information about routing protocols.

Network Address Translation

Network Address Translation (NAT) is an extension of routing that allows the router to modify the addresses in the packets it forwards. Traditional routers examine addresses, but they don't change them. NAT boxes convert the IP addresses used on the local network to "official" IP addresses. This allows you to use a private network number and still have Internet access. The private network numbers defined in RFC 1918 are

- Network 10.0.0.0
- Networks 172.16.0.0 to 172.31.0.0 (Network 172.16.0.0 is used for the examples in this book)
- Networks 192.168.0.0 to 192.168.255.0

Private network numbers are popular, and for some good reasons:

- Using a private network number reduces paperwork. You don't have to ask anyone's permission to use these addresses. No applications, no fees. Just do it.
- The addresses are yours. If you change ISPs, there is no need to renumber the hosts on the network. You may need to change the configuration of the NAT box, but that is probably easier than changing the configuration of all of your desktop systems.
- You conserve IP addresses. Having more addresses than you really need can make designing a network much easier, but you don't want more than you need if you're wasting valuable IP addresses. When you use private addresses, you don't waste any IP addresses. These addresses are reuseable, and the same addresses you're using are probably being used by hundreds of other private networks around the world.
- Private IP addresses reduce address spoofing. *Spoofing* is a security attack in which someone at a remote location pretends to be on your local network by using one of your network addresses. Private IP addresses should not be forwarded through the Internet, so spoofing one of these addresses won't do the attacker much good.

Private network numbers are explicitly defined for private use. They cannot be routed through the Internet because any number of private networks might be using the same addresses. Before packets originating from a host that uses a private IP address can be forwarded to an external network, the source address in the packet must be converted to a valid Internet address.

Weigh all of the factors before you decide to use NAT. Network address translation has some problems:

- It places a small additional overhead on the router, which reduces the router's performance.
- It doesn't work well with all protocols. TCP/IP protocols were not designed with NAT in mind.
- It interferes with end–to–end authentication schemes that authenticate the source address.

Linux 2.4 implements IP address translation in the kernel using the iptables command. Linux includes IP address translation as part of the firewall software that comes with the system. Firewalls and how to configure a Linux server as a firewall are discussed in Chapter 12, "Security." Chapter 12 provides the real details of the iptables command. This chapter looks at the one aspect of the iptables command that allows you to translate addresses.

Configuring a Linux NAT Server

Despite the fact that address translation is included in the packet–filtering software used to build a firewall, it is not specifically a security feature. A very common use for address translation is to connect a small network to the Internet. Assume that you have a small office network that connects to the Internet through a local ISP. Further, assume that the ISP assigns the office only one IP address, even though you have four computers on your network. Using a Linux NAT box, all four computers can communicate with the Internet with only one valid IP address.

The iptables command processes packets through sets of filtering rules. These sets of rules are called *chains*. The primary reason to construct these rules is security, and the chains that apply to security are described in Chapter 12. When iptables is used for address translation, which Linux

calls *masquerading*, two special rule chains are used:

- The *prerouting* chain that handles packets before the router processes the packet for routing
- The *postrouting* chain that handles packets after the router processes the packet for routing

The iptables command has three policies that are specific to address translation. Policies, sometimes called *targets* in the iptables syntax, are procedures that process packets. Several policies have been predefined for security processing, and three have been predefined for NAT processing:

MASQUERADE This is used to translate an address to the address assigned to the NAT box.

SNAT This is used to statically map one specific address to another specific address.

DNAT This is used to dynamically map an address to the next available address in a group of addresses.

Assume that the small office network attaches to a Linux router acting as a NAT box through the router's eth0 interface. Further, assume that the Linux router connects to the outside world through its eth1 interface, and that the address assigned to that interface is the official address provided by the ISP. The following iptables command will do the job:

```
# iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
```

The –t nat option tells iptables that it is working with the address translation chains. The –A option adds this rule to the POSTROUTING chain, which means that this translation will happen after the router determines how to deliver the packet. The –o eth1 option tells iptables that the rule should be applied to any packet that is heading out of the eth1 interface. (–i is used for packets coming in an interface.) The –j MASQUERADE option tells iptables that when a packet matches the conditions of having been processed for routing and of being routed out of interface eth1, iptables should jump to the MASQUERADE policy to complete processing. As noted earlier, the MASQUERADE policy converts the source address of the packet to the address assigned to the router—in this case, it converts the source address to the address assigned to interface eth1.

With this rule, the one official address assigned to the router can be used to connect all four of the systems on our small network to the Internet at large. Combined with the Linux routing features discussed earlier, when the NAT rule is installed, Linux completely meets the routing needs of our small office network.

In Sum

This chapter concludes the Internet server operations part of this book. All of the basic services needed for an Internet server have been covered: remote login, file transfer, e–mail, web service, and routing.

The next chapter begins Part Three, "Departmental Server Configuration," in which the services needed on a departmental server are examined. To begin, Chapter 8 covers the DHCP server that simplifies the configuration of desktop computers.

Part III: Departmental Server Configuration

Chapter List

Chapter 8: Desktop Configuration Servers
Chapter 9: File Sharing
Chapter 10: Printer Services
Chapter 11: More Mail Services

Part Overview

Featuring:

- Configuring a DHCP server, client, and relay agent
- Understanding Linux file permissions
- Configuring an NFS server and client
- Using mount, fstab, and automounter to access filesystems
- Understanding NetBIOS and SMB
- Configuring an SMB server or client with Samba
- Installing, configuring, and sharing printers
- Creating a mailbox server with POP or IMAP
- Controlling spam e−mail
- Filtering e−mail with procmail

Chapter 8: Desktop Configuration Servers

Overview

TCP/IP is able to link the world together into a global Internet because it does not depend on any one physical network technology. It can run over the modem attached to a PC or over the fiber–optic network attached to a super computer. It does this by creating a logical network on top of the physical networks that is independent of the specific characteristic of any one network.

However, this flexibility comes at the price of complexity. It is more difficult to configure a computer to run TCP/IP than it is to configure it for some other networks.

You're a technical person—that's why you run the network. Configuring TCP/IP may seem very simple to you, but it can be a daunting task for the average user setting up a PC. If your network is small, you can manually configure all of the desktop systems yourself. On a large network, manual configuration becomes an impossible task. Even on a small network, fixing the configuration every time a user upgrades is a thankless and boring job. The solution is to create a server that does this job for you, which is the topic of this chapter.

Understanding Configuration Protocols

Protocol developers have worked to reduce the burden of manual system configuring for a long time. Some of the documents that define the configuration protocols are more than 15 years old. Surprisingly, these protocols have come into widespread use only in the past few years. This is partly because the early users of the Internet were technical people who liked to configure their own systems, and partly because of the tremendous growth in the number of systems running TCP/IP that occurred at the end of the 1990s. Microsoft also deserves some credit for pushing hard to get people to use *Dynamic Host Configuration Protocol* (DHCP), which is the best of the configuration protocols. This section examines DHCP, as well as the other configuration protocols used to configure desktop systems.

Bootstrap Protocol

Bootstrap Protocol (BootP) was the first comprehensive configuration protocol. It can provide all of the information commonly used to configure TCP/IP—from the client's IP address to which print server the client should use. The BootP protocol is designed to deliver this information to the client, even though the client doesn't have an IP address.

Here's how it works. The BootP client broadcasts a BOOTREQUEST packet to UDP port 67, using a special IP broadcast address of 255.255.255.255 that is called the limited broadcast address. The broadcast address assigned in Chapter 2, "The Network Interface," with the ifconfig command was made up of the network address with a host field of all ones; for example, 172.16.55.255. Clearly, a BootP client that doesn't know the network address couldn't use such a broadcast address, which is why the limited broadcast address is used.

Note Unless specially configured to do so, routers do not forward the limited broadcast address. For this reason, configuration servers are traditionally departmental servers, with one server placed on each subnet. Later in this chapter, we will see how relay servers can be used to support a centralized configuration server for organizations that prefer centralization over distributed departmental servers.

The client puts all of the information it knows about itself in the BOOTREQUEST packet, which might be only its physical–layer address. When a BootP server receives a packet on port 67, it creates a BOOTREPLY packet by filling in as much of the missing configuration information as it can. The server then broadcasts the packet back to the network using UDP port 68. The client listens on port 68. When it receives a packet on the port that contains its physical–layer address, it uses the information from the packet to configure TCP/IP.

Dynamic Host Configuration Protocol

BootP is simple and effective. So effective, in fact, that it became the basis for Dynamic Host Configuration Protocol. DHCP operates over the same UDP ports, 67 and 68, as BootP. It provides all of the services of BootP as well as some important extensions.

DHCP is designed to provide all possible TCP/IP configuration parameters to its clients. DHCP includes every parameter that was defined in the "Requirements for Internet Hosts" RFC, which means that everything necessary for TCP/IP can be configured from the DHCP server.

The other, and probably more important, extension is that DHCP permits IP addresses to be dynamically assigned. Manually configuring a system with ifconfig permanently assigns an address to the interface. The address assigned to a host through BootP is also a permanent assignment. Both of these techniques are *static*—the address is permanently assigned and cannot be used for any other host. With DHCP, an address is "leased" to the host for a specific period of time. When the time expires, the host must either renew the lease or return the address to the server so that it can be assigned to another system.

The advantages of dynamic address assignment are as follows:

- You don't need to create a custom configuration for each host. On a BootP server, you must create a configuration for every client because you are responsible for assigning each system a unique IP address. With a DHCP server, the server is responsible for the address assignments.
- It makes more effective use of scarce IP addresses. Unused addresses are returned to the address pool, from where they can be used again.

Dynamic address assignment, like everything else, is not perfect. DNS may not know about addresses that are assigned through DHCP, which means that remote computers could not look up the address of a system that got its address from DHCP. In that case, a system using a dynamically assigned address could not offer services to other systems.

This is a shortcoming, but it's not a fatal flaw. First, only officially recognized servers should offer services to other systems, and skilled technical people who don't have any trouble doing a TCP/IP configuration run most servers. Second, the number of servers on a network is small compared with the total number of systems, and therefore the burden of server configuration is correspondingly small. Third, the bulk of the systems on a network are desktop systems that shouldn't offer TCP/IP services to remote users, so they do not need static IP addresses. This makes desktop clients perfect candidates for dynamic address assignment. And finally, there are techniques for coordinating addresses between DHCP and DNS using *Dynamic DNS* (DDNS). To find out more about DDNS, see *Linux DNS Server Administration* by Craig Hunt, Sybex 2001. Even without DDNS, DHCP works fine for most systems, and can dramatically decrease the configuration workload.

Reverse Address Resolution Protocol

Before leaving the topic of configuration protocols, we should quickly mention *Reverse Address Resolution Protocol* (RARP). As the name implies, it is the reverse of ARP. Instead of asking for an Ethernet address in response to an IP addresses, this protocol broadcasts an Ethernet address, and asks for an IP address in response.

A RARP server uses the /etc/ethers file to map Ethernet addresses to IP addresses. It then sends the IP address from the ethers file to the client system. A sample /etc/ethers file is shown here:

```
00:00:C0:4F:3E:DD bluejay
00:10:4B:87:D4:A8 duck
08:00:20:82:D5:1D raven
00:00:0C:43:8D:FB osprey
```

Each line in the file contains an Ethernet address, followed by a hostname or IP address. Hostnames are most commonly used, but they must be valid names that map to IP addresses.

We mention this protocol because the nsswitch.conf file covered in Chapter 4, "Linux Name Services," includes /etc/ethers as a part of the NIS service, which might make you curious about it. However, you should not use RARP. RARP only provides the client with an IP address. No other configuration information is provided. Much better configuration servers are available for Linux, including DHCP, which is the right configuration server for most networks.

Installing the DHCP Server

Many Linux distributions include the DHCP daemon (dhcpd). dhcpd is the server side of DHCP, and is required only on the DHCP server. The clients do not run dhcpd. Information on configuring a DHCP client is provided later in the chapter.

The DHCP server software is a component that can be selected during the initial Linux installation. If the DHCP server software was not installed as part of the initial installation, install it now using a package manager such as the rpm command or the X Windows tool gnorpm. If you use gnorpm, DHCP can be found in the System Environment/Daemons display.

Note For an example of using the rpm command to install a software package, see the "Using Package Manager" section in Chapter 5, "The Apache Web Server."

Of course, it is possible that your Linux system does not come with the DHCP software, or that you want a more recent version than the one that comes with your system. In either case, you can download the source code for dhcpd from http://www.isc.org/ or (via anonymous FTP) from ftp.isc.org/isc/dhcp , where it is stored in a gzipped tar file. Restore the tar file, change to the directory it creates, and run the ./configure script located there. configure determines the type of system you're running, and creates the correct Makefile for that system. Run make to compile the software. (If you have an outdated Linux system, see the following sidebar, "Using dhcpd with Old Linux Kernels," for information about problems that you might encounter.) On current Linux systems, dhcpd should compile without errors. Of course, things may change with future releases. If you get errors, send mail to the dhcp–server@isc.org mailing list, describing your configuration and the exact problem you have. The list is read by most of the people using dhcpd, and someone may have already solved your problem. To join the mailing list, go to www.isc.org/services/public/lists/dhcp–lists.html and fill out the form.

dhcpd works well with the Linux 2.4 kernel. However, if your Linux system uses an outdated kernel, installing dhcpd may not be all that is required to get it running. In that case, there are a few potential problems that need to be addressed by system–specific configurations. The best way to solve these problems is to update to a new kernel. If, for some reason, you are unwilling to update your kernel, you should read this sidebar.

If you provide service to Microsoft Windows DHCP clients, you may encounter problems with the limited broadcast address. If it appears that Microsoft Windows clients do not see DHCP messages from the server while other types of clients do, you need to define a specific route for the limited broadcast address on your Linux server. To do so, first add the name all–ones to the /etc/hosts table:

```
255.255.255.255          all-ones
```

Then, add a route for the limited broadcast address:

```
route add -host all-ones dev eth0
```

To reinstall the special route after each boot, add the route statement to the rc.local startup script. Old distributions with the versions of the kernel that require it often include the code to add the limited broadcast route in the /etc/rc.d/init.d/dhcpd script used to start dhcpd.

In addition to the limited broadcast problem, there are some other problems that relate to old Linux kernels.

Multiple network interfaces dhcpd cannot use multiple interfaces with Linux kernels prior to version 2.0.31.

SO_ATTACH_FILTER undeclared This error may occur when compiling dhcpd under Linux 2.2. If it occurs, the symbolic link /usr/include/asm may be broken. That link should point to the Linux asm headers.

Protocol not configured To run under Linux 2.1 and 2.2, dhcpd needs the CONFIG_PACKET and CONF_FILTER options configured in the kernel. If the message Set CONFIG_PACKET=y and CONFIG_FILTER=y in your kernel configuration is displayed during the dhcpd build, you need to reconfigure the kernel and enable these options. See Chapter 13, "Troubleshooting," for information on configuring the kernel.

IP BootP agent Linux 2.1 will run dhcpd only if the BootP agent is enabled. (The BootP agent is part of dhcpd.) Check to see whether /proc/sys/net/ipv4/ip_bootp_agent exists. If it does, check to see if it contains a 1. If it doesn't exist or contain a 1, insert the following line into a startup script to write a 1 to the ip_bootp_agent file:

```
echo 1 > /proc/sys/net/ipv4/ip_bootp_agent
```

Linux kernel 2.4 solves all of these problems. However, as the Linux kernel versions change and new releases of dhcpd are issued, new problems may emerge. See the README file that comes with the dhcpd distribution for the latest information.

Running *dhcpd*

Under Red Hat Linux 7.2, the DHCP daemon is started by the /etc/init.d/dhcpd script. The script accepts the same arguments as the /etc/init.d/named script described in Chapter 4. The most commonly used arguments are start, stop, and restart. For example, the following command will start dhcpd:

```
[root]# service dhcpd start
Starting dhcpd: [  OK  ]
```

Use a tool such as chkconfig or tksysv to ensure that the /etc/init.d/dhcpd startup script is run whenever the system reboots. On our sample Red Hat system, the following chkconfig commands cause the dhcpd script to run whenever the system reboots in runlevel 3 or 5:

```
[root]# chkconfig --list dhcpd
dhcpd           0:off   1:off   2:off   3:off   4:off   5:off   6:off
[root]# chkconfig --level 35 dhcpd on
[root]# chkconfig --list dhcpd
dhcpd           0:off   1:off   2:off   3:on    4:off   5:on    6:off
```

The /etc/init.d/dhcpd script uses the dhcpd command to start the DHCP server. The syntax of the dhcpd command is

dhcpd [**-p** ***port***] [**-f**] [**-d**] [**-q**] [**-cf** ***file***] [**-lf** ***file***] [***interface-list***]

-p *port* Defines an alternate port. Normally, dhcpd listens for client requests on port 67, and responds on port 68. Use the -p option to change to non-standard ports. This is used only for testing.

-f Runs dhcpd as a foreground process. This is used only for debugging.

-d Sends error messages to stderr instead of to syslogd.

-q Prevents dhcpd from printing out its startup message.

-cf *file* Identifies the file from which dhcpd should read its configuration. By default, dhcpd reads its configuration from /etc/dhcpd.conf.

-lf *file* Identifies the file to which dhcpd should write address lease information. By default, dhcpd writes lease information to /var/lib/dhcp/dhcpd.leases.

interface-list Lists the names of the interfaces that dhcpd should monitor for client request. By default, dhcpd listens to all interfaces that support broadcasts.

To set dhcpd command-line arguments on a Red Hat system, store the arguments in the file /etc/sysconfig/dhcpd. The /etc/init.d/dhcpd script reads arguments from that file before starting the DHCP daemon. For example, to make dhcpd read its configuration from a file named /var/dhcp/test.conf, create a /etc/sysconfig/dhcpd file that contains the following:

```
[root]# cat /etc/sysconfig/dhcpd
# Command line options here
DHCPDARGS=-cf /var/dhcp/test.conf
```

This causes dhcpd to read its configuration from /var/dhcp/test.conf, but the /etc/dhcpd.conf file is still required! The /etc/init.d/dhcpd script does not run dhcpd unless it finds both the /etc/ dhcpd.conf file and the /var/lib/dhcp/dhcpd.leases file.

Initializing the *dhcpd.leases* File

dhcpd stores a database of the address leases it has assigned in the dhcpd.leases file. The file must exist for dhcpd to boot. When the server is first installed, create an empty dhcpd.leases file to ensure that the daemon starts correctly.

dhcpd writes database entries into the file as ASCII text. If you're curious, you can view the file to see what leases have been assigned. Entries in the file have the following format:

```
lease address {statements}
```

Each lease begins with the keyword lease and the IP address that is assigned by the lease. This is followed by a group of statements that define the characteristics of the lease. Possible values that might appear in the list of statements include the following:

starts *date* Records the start time of the lease. *date* contains the weekday, year, month, day, hour, minute, and second when the lease started.

ends *date* Records the time when the lease will end. *date* contains the weekday, year, month, day, hour, minute, and second when the lease will end.

hardware *hardware−type mac−address* Records the client's physical−layer address. On an Ethernet, the *hardware−type* is ethernet, and the *mac−address* is the Ethernet address.

uid *client−identifier* Records the client's DHCP identifier if one was used by the client when it obtained the lease. Most clients are identified by their MAC addresses, and do not require a separate DHCP identifier.

client−hostname "*name*" Records the client's hostname if the client provided it using the client−hostname DHCP option. (Much more will be said about DHCP options later in this chapter.)

hostname "*name*" Records the client's hostname if the client provided a hostname using the hostname DHCP option. Microsoft Windows clients send their hostnames to the server using the hostname option.

abandoned Identifies this as an abandoned lease. If the server has trouble assigning an address (either because the client rejects the address, or the server determines that an unassigned address is already in use), the server marks it "abandoned" until it runs short of available addresses and needs to try this one again.

Use this information when you want to examine the contents of the dhcpd.leases file. Beyond that, you don't need to be concerned about these commands. When you create the file, you create it empty. After you create it, you can forget about it because dhcpd maintains the file. The file that does require your input, however, is the dhcpd.conf file that is used to configure the server.

Configuring the DHCP Server

After dhcpd is installed, it must be configured. The DHCP daemon is configured through the dhcpd.conf file. The file can contain an extensive list of configuration commands that provide direction to the server and configuration information to the clients.

A DHCP server can be configured to provide service to individual hosts and to entire subnets of hosts. The dhcpd configuration language includes host and subnet statements that identify the scope of systems being serviced.

A host statement might contain the following:

```
host osprey {
  hardware ethernet 00:00:0C:43:8D:FB ;
  fixed-address 172.16.70.8 ; }
```

This statement defines the hostname, Ethernet address, and IP address of the client. When configured in this way, dhcpd provides a static address assignment similar to the service provided by a BootP server. This statement matches the client's Ethernet address against the configuration entry, and returns a static IP address to the client. Using this technique, DHCP can be used to assign static addresses to systems, such as name servers, that require static addresses.

The format of the subnet statement is

```
subnet 172.16.70.0 netmask 255.255.255.0 {
  range 172.16.70.100 172.16.70.250 ; }
```

The subnet statement declares that the system is providing DHCP service to network 172.16.70.0. Furthermore, the range clause says that the server is providing dynamic addressing, and that the range of addresses available for dynamic allocation is from 172.16.70.100 to 172.16.70.250. The range clause defines the scope of addresses available for dynamic assignment. It is always associated with a subnet statement, and the range of addresses defined in the clause must fall within the address space of the subnet.

Controlling Server and Protocol Operations

Configuration parameters and options can be associated with individual host and subnet statements. The group statement can be used to apply parameters and options to a group of host or subnet statements. Additionally, configuration parameters and options can be specified that apply to every system and network defined in the configuration file. With this flexibility, the dhcpd configuration language allows you to create every conceivable configuration.

The configuration language includes several things that control the operation of the DHCP server and the DHCP protocol. First are the allow and deny statements that control how dhcpd handles certain client requests. Each statement begins with either the command allow or deny, followed by a keyword that describes the request that is being allowed or denied. The three possible keywords are the following:

> **unknown−clients** Configuration requests from clients for which the server does not have a specific host entry can be allowed or denied. By default, unknown clients are allowed. If they are denied, some of the dynamic capabilities of DHCP are lost.

bootp The server can be directed to handle configuration requests from BootP clients or to ignore them. By default, BootP clients are allowed so that all clients, BootP and DHCP, can be handled by a single server.

booting The deny booting statement is used within a host statement to tell the server that it should not handle configuration requests from a specific client. The default is to handle a client's configuration request, which means that it is not necessary to use the allow booting statement.

The authoritative or not authoritative statements tell the server whether or not the configuration information it has is known to be accurate. By default, a DHCP server is authoritative, and the configuration information it provides is accurate. If, for some reason, a DHCP server is configured by someone who does not have authority over the network configuration, the not authoritative statement can be used to limit the amount of authority the server asserts over the clients. Avoid this. Only authoritative DHCP servers should be set up—and only by the network administrator.

In addition to the various statements described previously, there are several configuration parameters that control server and protocol operation:

always–reply–rfc1048 *flag* Some older BootP devices may only be able to accept responses formatted according to RFC 1048. If that is the case, set *flag* to true, and place this parameter in a host statement to send responses to a specific BootP client in the old format.

default–lease–time *seconds* Defines the length of time given to an address lease if the client does not request a specific lease length.

dynamic–bootp–lease–cutoff *date* Defines a termination date for addresses assigned to BootP clients. By default, dhcpd assigns permanent addresses to BootP clients. This parameter changes that behavior, but it cannot change the way that BootP clients work. If the lease runs out, a BootP client will not know that its address is invalid.

dynamic–bootp–lease–length *seconds* Defines the maximum length of an address lease, in seconds, for a BootP client. Because BootP clients do not renew address leases, a client that does not boot frequently enough will lose its lease.

filename "*file*" Defines the pathname of the boot file for diskless clients.

fixed–address *address*[, *address*...] Assigns a permanent IP address to a host as part of a host statement. More than one address can be supplied for a client that boots on more than one subnet.

get–lease–hostnames *flag* If *flag* is true, dhcpd does a reverse lookup for every dynamically assigned address, and sends the hostname it gets from DNS to the client. This process can add lots of overhead for the server on a big network. By default, *flag* is false, and no lookups are done.

hardware *hardware–type mac–address* Defines the client's Ethernet address. The hardware parameter must be part of a host statement, which uses the physical–layer address to tie the host information to a specific client. The physical–layer address is the only way for a BootP client to be recognized. DHCP clients can use other values

in addition to the physical–layer address to identify themselves.

max–lease–time *seconds* Defines the maximum length of a lease, regardless of the lease length requested by the client. The length of time is defined in seconds.

next–server *name* Defines the hostname of the server from which the boot file is to be loaded. This is significant only for diskless clients that boot from a server.

range [dynamic–bootp] *low–address* [*high–address*] Defines the scope of addresses available for dynamic assignment. The keyword dynamic–bootp, if included on the range statement, tells dhcpd to assign dynamic addresses to BootP clients as well as DHCP clients. Because BootP clients do not understand address leases, they are normally not given dynamic addresses.

server–identifier *address* Defines the IP address of the server that is sent to clients. By default, the address of the server's network interface is used. Set this value only if, for some reason, the server gets the wrong address from its configuration.

server–name "*name*" Defines the server name that is sent to clients.

use–host–decl–names *flag* Tells dhcpd to send the name provided on the host statement to the client as the client's hostname if *flag* is true.

use–lease–addr–for–default–route *flag* When *flag* is true, sends the client its own address as the default route instead of sending the true default route. This forces Windows 95 clients to use ARP for all addresses, which means that the real router must be configured as a proxy ARP server.

These statements and parameters provide configuration information to the server, defining how the server operates. However, most of the information in the configuration file is information for the client. The next section looks at the configuration options that provide configuration information to the clients.

dhcpd Configuration Options

The dhcpd option statements cover all DHCP configuration options defined in the RFCs. Furthermore, any new option that might be defined in the future can be included in the dhcpd configuration by using the decimal option code assigned to it in the RFC that describes the option. An option name in the form option–*nnn*—where *nnn* is the decimal option code—can be used to add any new option to the dhcpd.conf file. For example, assume that you want to assign the string "yes" to a new DHCP configuration option that has an option code of 142. You could add the following to your configuration file:

```
option option-142 "yes"
```

The statement begins with the keyword option, which is followed by the name of the option and the value assigned to the option. In the example, the name is option–142, and the value is "yes".

There are more than 60 standard DHCP configuration options. To make the following list of options more manageable, it is divided into six sections: Basic Options, Tuning Options, Routing Options, NetBIOS Options, Diskless Client Options, and Other Server Options.

Basic Options

The basic options define such things as the address, the subnet mask, and the default router. The following list also includes the servers that are most likely to be included in an average configuration, such as DNS servers and print servers.

option broadcast−address *address* Defines the broadcast address.

option dhcp−client−identifier *string* Defines the string used to identify a DHCP client in lieu of the client's hardware address.

option domain−name *domain* Defines the domain name.

option domain−name−servers *address−list* Lists the addresses of the DNS name servers.

option host−name *host* Defines the client's hostname.

option lpr−servers *address−list* Lists the addresses of the print servers.

option nntp−server *address−list* Lists the addresses of the Network News Transfer Protocol (NNTP) servers. These are the servers from which the client gets news service.

option ntp−servers *address−list* Lists the addresses of the Network Time Protocol (NTP) servers.

option pop−server *address−list* Lists the addresses of the POP3 mailbox servers.

option routers *address−list* Lists the routers on the client's subnet in order of precedence.

option smtp−server *address−list* Lists the addresses of the SMTP e−mail servers.

option subnet−mask *mask* Defines the subnet mask. If this option is not defined, the network mask from the subnet statement is used.

option time−offset *seconds* Defines the offset from Coordinated Universal Time (UTC) of this time zone.

Tuning Options

The following list identifies the options that are used to tune the TCP/IP protocol.

option all−subnets−local 0 | 1 Specifies whether all subnets use the same Maximum Transmission Unit (MTU). 1 means that they all do, and 0 means that some subnets have smaller MTUs.

option arp−cache−timeout *seconds* Defines how long clients should cache ARP table entries.

option default−ip−ttl *ttl* Defines the default time−to−live (TTL) for outgoing

datagrams. Possible values are 1 to 255.

option default−tcp−ttl *ttl* Defines the default TTL value for TCP segments. Possible values are 1 to 255.

option ieee802−3−encapsulation 0 | 1 0 tells the client to use Ethernet II (DIX) Ethernet encapsulation, and 1 tells the client to use IEEE 802.3 encapsulation.

option interface−mtu *bytes* Defines the Maximum Transmission Unit (MTU) that should be used by the client.

option mask−supplier 0 | 1 Specifies whether the client should respond to ICMP subnet mask requests. 0 means no, and 1 means yes. 0 is the default because clients do not normally respond to subnet mask requests.

option max−dgram−reassembly *bytes* Defines the largest datagram the client must reassemble. It cannot be less than 576 bytes.

option path−mtu−aging−timeout *seconds* Sets the number of seconds for timing out RFC 1191 Path MTU values.

option path−mtu−plateau−table *bytes*[, *bytes*...] Defines a table of MTU sizes for RFC 1191 Path MTU Discovery. The minimum MTU value is 68.

option perform−mask−discovery 0 | 1 0 enables ICMP mask discovery, and 1 disables it. Because the DHCP server provides the correct subnet mask, ICMP mask discovery is rarely used on networks that have a DHCP server.

option tcp−keepalive−garbage 0 | 1 Specifies whether the TCP keepalive messages should include an octet of garbage for compatibility with older implementations. 0 means don't send a garbage octet, and 1 means send it. Keepalives are generally discouraged.

option tcp−keepalive−interval *seconds* Defines the number of seconds TCP should wait before sending a keepalive message. Zero (0) means that TCP should not generate keepalive messages. Keepalive messages are generally discouraged.

option trailer−encapsulation 0 | 1 0 means the client should not use trailer encapsulation, and 1 means the client should use trailer encapsulation.

Routing Options

The following options all relate to routing. Except for defining the default route, everything related to routing is covered here. The default route is listed under "Basic Options."

option ip−forwarding 0 | 1 0 tells the client to disable IP forwarding, and 1 says to enable it.

option non−local−source−routing 0 | 1 0 tells the client to disable non−local source routing, and 1 says to enable it. 0 is more secure. Source routes are a potential security problem that can allow intruders to route data off of the local network.

option policy−filter *address/mask* Lists the only valid destination/mask pairs for incoming source routes. Any source−routed datagram whose next−hop address does not match one of the filters is discarded by the client.

option router−discovery 0 | 1 1 means the client should locate routers with the RFC 1256 Router Discovery mechanism, and 0 means it shouldn't. Because the DHCP server can provide the correct list of routers, router discovery is not commonly used on networks that have a DHCP server.

option router−solicitation−address *address* Defines the address to which the client should send RFC 1256 Router Discovery requests.

option static−routes *destination gateway*[,...] Defines static routes for the client as a list of destination and gateway pairs.

NetBIOS Options

The following options configure NetBIOS over TCP/IP.

option netbios−dd−server *address−list* Lists the addresses of the NetBIOS datagram distribution servers (NBDDs).

option netbios−name−servers *address−list* Lists the IP addresses of the NetBIOS name servers (NBNSs).

option netbios−node−type *type* Defines the NetBIOS node type of the client. A type of 1 is a NetBIOS B−node, 2 is a P−node, 4 is an M−node, and 8 is an H−node.

option netbios−scope *string* Defines the NetBIOS over TCP/IP scope parameter.

Note To understand these values, see the discussion of NetBIOS in Chapter 9, "File Sharing."

Diskless Client Options

DHCP can be used to boot diskless clients, such as X terminals. This list contains the options that pertain to diskless clients.

option bootfile−name *string* Identifies the client's boot file.

option boot−size *blocks* Defines the number of 512−octet blocks in the boot file.

option merit−dump *path* Defines the pathname of the file the client should dump core to in the event of a crash.

option root−path *path* Defines the pathname of the client's root disk.

option swap−server *address* Defines the IP address of the client's swap server.

Other Server Options

The most commonly used network servers are covered in the "Basic Options" list, but there are several other network servers available. Options relating to those servers are listed here.

option cookie−servers *address−list* Lists the addresses of the RFC 865 cookie servers.

option finger−server *address−list* Lists the finger servers available to the client. finger servers are used at sites that block finger traffic at the firewall.

option font−servers *address−list* Lists the X Windows font servers.

option ien116−name−servers *address−list* Lists the IEN 116 name servers. IEN 116 is an obsolete name service.

option impress−servers *address−list* Lists the addresses of the Image Impress servers.

option irc−server *address−list* Lists the addresses of the Internet Relay Chat (IRC) servers.

option log−servers *address−list* Lists the MIT−LCS UDP log servers.

option mobile−ip−home−agent *address−list* Lists the Mobile IP home agents the client should use.

option nis−domain *name* Defines the name of the client's Network Information Services (NIS) domain.

option nis−servers *address−list* Lists the addresses of the NIS servers.

option nisplus−domain *name* Defines the name of the client's NIS+ domain.

option nisplus−servers *address−list* Lists the IP addresses of the NIS+ servers.

option resource−location−servers *address−list* Lists the addresses of the Resource Location servers.

option streettalk−directory−assistance−server *address−list* Lists the addresses of the StreetTalk Directory servers.

option streettalk−server *address−list* Lists the addresses of the StreetTalk servers.

option tftp−server−name *string* Oddly enough, *string* should be the hostname of the DHCP server because some clients use this option instead of server−name to identify the DHCP server.

option time−servers *address−list* Lists the addresses of the time servers.

option www−server *address−list* Lists the web servers available to the client. This is primarily useful for defining proxy web servers that a client must use.

option x−display−manager *address−list* Lists the X Windows display manager servers.

These six lists comprise quite a slew of possible options, most of which you will never use—not because the options are unimportant, but because the default values that most systems configure for these options are correct. However, this large set of configuration options permits you to control the complete TCP/IP system configuration from the DHCP server. Options, parameters, group statements, host statements, and subnet statements are combined together in the dhcpd.conf file to create the server configuration and to define the information sent to the clients. The next section examines a configuration file in detail.

Creating a *dhcpd.conf* File

dhcpd reads its configuration from the /etc/dhcpd.conf file. The dhcpd.conf file identifies the clients to the server, and defines the configuration that the server provides each client. The sample dhcpd.conf file shown in Listing 8.1 dynamically assigns IP addresses to the DHCP clients on a subnet, and supports a few clients that require static addresses.

Listing 8.1: A Sample *dhcpd.conf* File

```
# Define global values that apply to all systems.
max-lease-time 604800;
default-lease-time 86400;
option subnet-mask 255.255.255.0;
option domain "foobirds.org";
option domain-name-servers 172.16.55.1, 172.16.5.1;
option pop-server 172.16.18.1;

# Define the dynamic address range for the subnet.
subnet  172.16.55.0 netmask 255.255.255.0 {
    option routers 172.16.55.1;
    option broadcast-address 172.16.55.255;
    range 172.16.55.64 172.16.55.192;
    range 172.16.55.200 172.16.55.250;
}

# Use host statements for clients that get static addresses
group {
    use-host-decl-names true;
    host kestrel {
        hardware ethernet 00:80:c7:aa:a8:04;
        fixed-address 172.16.55.4;
    }
    host ibis {
        hardware ethernet 00:00:c0:a1:5e:10;
        fixed-address 172.16.55.16;
}
```

The file begins with the parameters and options that apply to all of the subnets and clients served. The first two lines define how dhcpd should handle dynamic address assignments:

max-lease-time Specifies the longest address lease that dhcpd is allowed to grant, regardless of the lease length requested by the client. In the example, this parameter is one week.

default-lease-time Defines the address lease time used when a client does not request a specific address lease length. In the Listing 8.1 example, the default lease is set to one day (86400 seconds).

The meaning of the next four lines is easy to see. These options define the subnet mask, domain name, domain server addresses, and POP server address used by all clients.

The network that dhcpd serves is identified by an address and an address mask in the subnet statement. dhcpd provides configuration services only to clients that are attached to this network or identified directly by host statements. The options and parameters in the subnet statement apply only to the subnet and its clients. In the example, the options define the subnet's default router and the broadcast address.

The two addresses in the range parameter define the scope of addresses that are available for dynamic address allocation. The first address is the lowest address that can be automatically assigned, and the second address is the highest address that can be assigned. The subnet statement in the example has two range parameters to create two separate groups of dynamic addresses. This illustrates that you can define a noncontiguous dynamic address space with multiple range statements.

Note If a range parameter is defined in a subnet statement, any DHCP client on the subnet that requests an address is granted one, as long as addresses are available. If a range parameter is not defined, dynamic addressing is not enabled.

The configuration concludes with a group of host statements. The group statement that encloses the host statements applies the use−host−decl−name configuration parameter to all of the hosts. With this parameter set, the client at Ethernet address 00:80:c7:aa:a8:04 receives the hostname kestrel as part of its configuration information. Without this parameter, the client is sent an IP address, but is not sent a hostname. The Ethernet address contained in each host statement is the key used to identify the client and to determine which client receives what configuration information.

In the example, the host statements define the hostname and IP address of individual clients. The clients are assigned permanent addresses, so they do not need to renew leases on the addresses or even understand address leases. In addition to providing the information from the host statement to the clients, dhcpd sends them the subnet mask, domain name, DNS server addresses, and print server address defined in the global section of the configuration file.

Using the dhcpd.conf file, you can define any configuration information needed by any host or subnet your system serves. However, the DHCP server cannot respond to a request it does not receive. DHCP requests sent to the limited broadcast address might not be forwarded by your routers. To serve an enterprise network from a single server, you may need to install DHCP relay servers.

Configuring a *dhcrelay* Server

The DHCP relay agent (dhcrelay) is provided as part of the dhcpd distribution. The relay agent listens for DHCP boot requests, and forwards those requests to a DHCP server. The relay agent must be attached to the same subnet as the DHCP client because the request from the client uses the limited broadcast address. However, the relay does not need to share a subnet with the server because it uses the server's IP address to send the request directly to the server. The server then sends the DHCP reply packet back to the relay. The relay is responsible for broadcasting the reply packet on the local subnet so that the client can retrieve it.

Use the dhcrelay command to run a DHCP relay. A simple dhcrelay command is

```
dhcrelay -q 172.16.70.3
```

The –q option tells dhcrelay not to print out network configuration information when it starts. Normally, it does. If the dhcrelay command is placed in a startup script such as rc.local, use the –q option to prevent the configuration from printing out during the boot.

The IP address on the command line is the address of the DHCP server. When dhcrelay receives a DHCP request on the local network, it sends that request to 172.16.70.3. To use more than one DHCP server, specify multiple servers on the command line, as follows:

```
dhcrelay -q 172.16.70.3 172.16.90.4
```

dhcrelay sends the request to all of the servers listed on the command line.

On occasion, the DHCP relay service is configured on a Linux system that is also acting as a router for a small network. In that case, the router has more than one interface, and should be configured to provide DHCP relay service for the correct interface. For example, assume that you have a small router with two Ethernet interfaces: eth0 and eth1. eth0 is connected to a backbone network with other routers. eth1 is connected to a local subnet that has DHCP clients that need DHCP relay service. The dhcrelay command for this situation might be

```
dhcrelay -i eth1 172.16.70.3
```

This command tells dhcrelay to listen for DHCP requests only on interface eth1. When it receives any, it forwards them to 172.16.70.3.

The placement of DHCP servers and relays, and the coordination between all of these systems is an important part of planning a DHCP service. A conflict exists between centralizing control to reduce configuration errors, and improving booting efficiency and redundancy by placing the configuration servers close to the clients. The centralized solution places one large DHCP server in the central facility and a relay server on each subnet; the distributed solution uses no central server and places a DHCP server on every subnet. A real–life solution that combines elements of both is described in the following sidebar, "Placing DHCP Servers."

Placing DHCP Servers

An enterprise network composed of about 60 subnets decided to deploy DHCP servers. About 40 of the subnets were located at the sprawling headquarters facility. The remaining networks were located about 1500 miles away at a large production facility. The two sites were connected directly by a private leased circuit.

Clearly, the remote production facility could not depend on headquarters for configuration services. The company was forced to accept some level of distribution. Their computer services group offered two types of network support. For one level of support, they handled everything from setup to maintenance, and they used an internal billing mechanism to recover the cost of support from their internal customers. The other level of support allowed an organization to run its own subnet. The organization was assigned a subnet number, and was connected to the rest of the enterprise through a router run by the services group. This service was much less expensive. These two support models offered freedom when the organization could handle it, and offered support when the organization wanted and needed it.

Pleased with their support model, the company decided to replicate this service model in its configuration server architecture. Every organization that ran its own subnet was encouraged to buy

and install its own DHCP server; because those organizations handled their own maintenance and had the best idea of their own requirements, the final decision on using DHCP was left to them. Every subnet that was under central support was given a DHCP server. These servers were not quite central servers, and they were not quite distributed servers because each server was co–located with one of the routers run by the central services group. The DHCP server was then directly connected to each of the subnets coming into the router.

The 36 subnets that were under central support were covered by 10 Linux servers. Here's a sample configuration from one of those servers:

```
# Define global values that apply to all systems.

option subnet-mask 255.255.255.0;

option domain "foobirds.org";

option domain-name-servers 172.16.55.1, 172.16.12.3;


# Identify the subnets

subnet  172.16.42.0 netmask 255.255.255.0 {

    option routers 172.16.42.254;

    option broadcast-address 172.16.41.255;

    range 172.16.42.50 172.16.42.250;

}

subnet  172.16.52.0 netmask 255.255.255.0 {

    option routers 172.16.52.254;

    option broadcast-address 172.16.52.255;

    range 172.16.52.50 172.16.52.250;

}

subnet  172.16.62.0 netmask 255.255.255.0 {

    option routers 172.16.62.254;

    option broadcast-address 172.16.62.255;

    range 172.16.62.50 172.16.62.250;

}

subnet  172.16.72.0 netmask 255.255.255.0 {

    option routers 172.16.72.254;

    option broadcast-address 172.16.72.255;
```

```
    range 172.16.72.50 172.16.72.250;

}
```

No DHCP relay servers were required for this configuration. Sharing space with the routers made it possible to directly connect centrally maintained servers to every subnet. These "area" servers provided the advantages of central control with the speed and redundancy of distributed servers.

Configuring a DHCP Client

Not every Linux system is a server. It is also possible to configure a Linux desktop system as a DHCP client, and there are a few different tools available to do this. The dhcpd distribution provides dhclient to configure a desktop client. Most Linux systems ship with the DHCP Client daemon (dhcpcd), and Red Hat provides a client tool named pump. This section discusses all of these Linux DHCP clients, beginning with dhcpcd, because it is the most widely used.

Using the *dhcpcd* Client

As the name implies, the DHCP Client daemon (dhcpcd) provides the client side of the DHCP protocol exchange, and it provides the means for moving the information received from the DHCP server into the client's configuration. The syntax of the dhcpcd command is

```
dhcpcd [-dknrBCDHRT] [-t timeout] [-c filename] [-h hostname]
   [-i vendorClassID] [-I clientID] [-l leasetime] [-s [ipaddr]]
   [interface]
```

The name of the interface that dhcpcd should use for DHCP can be defined on the command line. An interface is specified only when the computer has more than one interface, and you want to use DHCP on only one of those interfaces.

The dhcpcd command accepts a large number of arguments. Two of these, –d and –T, are used for testing and debugging. The –T argument is *only* used for testing. It causes dhcpcd to run through the protocol exchanges with the server, but prevents it from reconfiguring the interface with the information it obtains. This allows you to check the server responses without affecting the client's configuration. The –d argument tells dhcpcd to log status messages via syslogd. These messages can be used for both monitoring and debugging.

A few of the command–line arguments impact the address lease. Two arguments, –k and –n, pass signals to the dhcpcd process. The dhcpcd –n command sends a SIGALRM signal that causes dhcpcd to renew the address lease. The command dhcpcd –k sends a SIGHUP signal to the dhcpcd process, which causes dhcpcd to release the current address lease, and delete the cached information about the lease. Because dhcpcd no longer has information about the address, it will not attempt to renew the lease on the address. Instead, it will negotiate with the server as if it had never been assigned an address. Although there is no command–line argument for the SIGTERM signal, dhcpcd does handle that signal. When dhcpcd receives a SIGTERM signal, it releases the address and terminates, but it does not remove the address from its cache. Therefore, when dhcpcd restarts after a SIGTERM, it attempts to renew its lease on the address.

Two of the arguments set timers relating to the address lease. The –t argument defines the maximum number of seconds that dhcpcd will wait for a server to provide a valid address. By default, it will wait 60 seconds. If dhcpcd gets an address before the timer expires, it exits with

status of 0; if it doesn't get an address it exits with a status of l to indicate failure.

The other timer argument is –l, which defines the number of seconds the client will request for an address lease lifetime. As you saw in the section on server configuration, the server does not have to grant the lease for the amount of time requested by the client. The server's max–lease–time parameter defines the upper time limit the server will allow. By default, the dhcpcd software asks for an infinite lease that, in effect, asks for the amount of time defined by max–lease–time.

Several of the command–line arguments control the protocol interactions. By default, dhcpcd is compliant with RFC 2131. The –r argument forces compliance with the older RFC 1541 specification to make dhcpcd backward–compatible with old servers. The –B argument causes the client to request that the server respond only via broadcasting. Normally, the server sends a broadcast response only when the client does not have an address. When an address is being renewed, the client uses that address during the negotiations, and the packets are all sent in both directions via unicast. –B asks the server to respond via broadcast, even if it could respond via unicast. –C tells dhcpcd to calculate a checksum for the packets it receives. Normally, checksums are done at the transport protocol level—by UDP in the case of DHCP—not at the application level. Finally, the –s option is used if the client will not accept an address from the server. A client that has its own permanent static address uses the –s argument to tell the server that address, and to request that the server send the client any other configuration information that the server can provide. Clearly, all of these arguments (–r, –B, –C, and –s) are used for exceptional cases. Average configurations do not need these arguments.

Three of the dhcpcd arguments, –h, –i and –I, are used to send information to the server that the server can use to identify the client. –h uses the DHCP hostname option to send a hostname string to servers that require it. Most server implementations do not require this information from clients. The –I argument defines the client identifier. By default, DHCP clients use their Ethernet MAC address as an identifier. The –I argument can be used to define some other identifier. However, if a special identifier is defined with –I, the same identifier must be defined on a dhcp–client–identifier option in the dhcpd.conf file on the server. Finally, the –i argument can be used to define a non–standard vendor identifier. By default, the vendor identifier is a combination of the operating system name and the machine type. For example, the vendor identifier for our sample Red Hat system is "Linux 2.4.7–10 i686."

There are also a few arguments that control the way dhcpcd uses the information it receives from the server. Normally, the client's hostname and domain name are set by the Linux hostname command very early in the startup process (see the discussion of the rc.sysinit script in Chapter 1), and dhcpcd does not override those values. Use –D to force dhcpcd to set the client's domain name from the DHCP domainname option supplied by the server. Use –H to cause dhcpcd to set the client's hostname using the hostname option supplied by the server.

Finally, the –R argument prevents dhcpcd from replacing the existing /etc/resolv.conf file with information it receives from the server. By default, dhcpcd overwrites any existing resolver configuration with the new configuration that the server provides. When the –R argument is specified, dhcpcd leaves the old /etc/resolv.conf file untouched, and writes the new resolv.conf file in the /etc/dhcpc directory.

dhcpcd stores the configuration information it receives in the following files in the /etc/dhcpc directory:

> **resolv.conf** This is a standard resolv.conf file created by dhcpcd from the search list and DNS server list received from the DHCP server. This file is created only in the

/etc/dhcpc directory when –R is specified on the dhcpcd command line. Normally, this file is written to the /etc directory.

dhcpcd–eth0.info This file contains the address lease information and other configuration options received from the server.

dhcpcd–eth0.cache This file contains data to be used in the DHCP packet when the client requests address renewal.

Notice that two of the filenames contain the device name eth0. This varies, based on the name of the interface on which the configuration information arrived. Most DHCP clients have only one network interface, so for most Linux clients, the filenames will be the ones listed here. Of these three files, dhcpcd–eth0.info is the most interesting. The resolv.conf file is usually a very simple resolver configuration that contains only name server addresses and a search list. The dhcpcd–eth0.cache file is unreadable, except for the string that defines the vendor identifier. On the other hand, the dhcpcd–eth0.info file is a human–readable ASCII file that contains the configuration values used by the client. Listing 8.2 shows the dhcpcd–eth0.info file from a Red Hat system.

Listing 8.2: A Sample *dhcpcd–eth0.info* File

```
[root]# ls /etc/dhcpc
dhcpcd-eth0.cache  dhcpcd-eth0.info
[root]# cat /etc/dhcpc/dhcpcd-eth0.info
IPADDR=172.16.0.3
NETMASK=255.255.255.0
NETWORK=172.16.0.0
BROADCAST=172.16.0.255
GATEWAY=172.16.0.1
HOSTNAME=dhcppc3
DOMAIN=tern.foobirds.org
DNS=172.16.5.1
DHCPSID=172.168.0.1
DHCPGIADDR=0.0.0.0
DHCPSIADDR=172.168.0.1
DHCPCHADDR=00:00:C0:9A:72:CA
DHCPSHADDR=00:A0:C5:E4:98:50
DHCPSNAME=sooty
LEASETIME=4294967295
RENEWALTIME=2147483647
REBINDTIME=3758096377
```

The sample file shown in Listing 8.2 starts with eight basic configuration values, including the client's IP address, the network mask, the broadcast address, the default gateway, the hostname, the domain name, and the address of the DNS server. This file can be incorporated in a shell script, and the values can be used inside that script to configure the system. The keyword value pairs used in this file are very similar to those used in the /etc/sysconfig/network–script/ifcfg–eth0 file mentioned in Chapter 2, "The Network Interface."

A Red Hat system with a static network configuration might contain an ifcfg–eth0 file such as the one shown in Listing 8.3.

Listing 8.3: A Sample *ifcfg–eth0* File

```
$ cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
```

```
ONBOOT=yes
BOOTPROTO=none
BROADCAST=172.16.12.255
NETWORK=172.16.12.0
NETMASK=255.255.255.0
IPADDR=172.16.12.2
USERCTL=no
```

Notice the BROADCAST, NETWORK, NETMASK, and IPADDR lines. These keywords exactly match keywords shown in Listing 8.2. In Listing 8.3, these values were statically defined by the system administrator. In Listing 8.2, the values came from the DHCP server. There are a few additional keyword/value pairs in Listing 8.3:

- DEVICE defines the device name, in this case eth0.
- ONBOOT specifies whether or not the interface is initialized when the system boots. Normally, an Ethernet interface is brought up and running every time the system boots.
- USERCTL specifies whether or not users can run usernetctl to bring the interface up or down. The usernetctl command is found on only a few versions of Linux. In this case, the value no prevents the user from downing the interface.
- BOOTPROTO identifies the configuration service used to configure the interface. In Listing 8.3, it is none, meaning that the interface is configured locally. Alternates are bootp if an old–fashioned BootP server is used or dhcp if a DHCP server is used.

The script /sbin/ifup performs the initial configuration of the network interface. The ifup script loads the keyword/value pairs from ifcfg–eth0. If BOOTPROTO is set to either dhcp or bootp, the script runs dhcpcd to update the network configuration values. If dhcpcd is not installed on the system, the script will run pump to update the network configuration.

Using the *pump* DHCP Client

pump is available on Red Hat systems, and supports both BootP and DHCP. Red Hat Linux 7.2 runs the pump command from the /sbin/ifup script only when dhcpcd is not found. This is an either/or proposition. You use either dhcpcd or pump; you do not use both. An interface that is configured by dhcpcd cannot be managed by pump. If your system uses dhcpcd, and most do, you can skip this section.

The pump command is very simple: pump –i eth0 configures interface eth0 with the information received from the DHCP server. Although pump normally runs from a startup script, it is possible to manually enter the command in order to check the status, release the address lease, or renew the lease. The possible pump command–line options are listed in Table 8.1.

Table 8.1: pump Command–Line Options

Command Option	Purpose
––config–file=*path*	Specifies the path to the pump configuration file. The default is /etc/pump.conf.
––help	Displays a help message for the pump command.
––hostname=*hostname*	Defines the hostname the client sends to the server.
––interface=*device*	Identifies the device being configured.
––kill	Kills the pump daemon.
––lease=*hours*	Requests a specific lease time in hours.

--leasesecs=*seconds*	Requests a specific lease time in seconds.
--lookup-hostname	Causes pump to look up the hostname via DNS.
--no-dns	Prevents pump from overwriting the /etc/resolv.conf file.
--no-gateway	Prevents pump from setting the client's default gateway.
--release	Releases the address assigned to the interface.
--renew	Requests a lease renewal.
--status	Displays the interface status.
--usage	Displays the syntax of the pump command.
--win-client-id	Formats the client identifier to be compatible with a Windows server.

Using the options in Table 8.1, you could query the status of an interface configured by pump by entering **pump -i eth0 --status.**

The pump.conf file is not required for an average configuration. pump.conf is used to customize the behavior of the pump program. The following directives can be used in the pump.conf file:

device *name* Identifies a network interface, such as eth0. This line is followed by a series of directives enclosed in curly braces ({}) that apply only to the network device identified by *name*.

domainsearch *search-list* Defines the DNS search list for this client. The search list received from the server is ignored, and this list is used.

nonisdomain Tells pump not to set a NIS domain—even if one is provided by the server, and no valid NIS domain is defined on the client. Normally, pump sets the NIS domain using the value supplied by the server if the client does not already have a valid NIS domain.

nogateway Tells pump not to set a default gateway for the client.

retries *count* Defines the number of times that pump should retry when negotiations with the server fail.

timeout *seconds* Defines the number of seconds that pump should wait for the negotiations with the server to conclude.

script *filename* Identifies a script that pump should run when a new address lease is obtained, when an address lease is successfully renewed, and when the interface is brought down. When a new address lease is obtained, pump runs the script, passing it the keyword up, the interface name, and the new IP address. When an address lease is successfully renewed, pump runs the script; and passes it the keyword renewal, the interface name, and the IP address. When the interface is brought down, pump releases the address lease, and runs the script, passing the script the keyword down and the interface name.

A simple pump.conf file is shown in Listing 8.4.

Listing 8.4: A Sample *pump.conf* File

```
$ cat /etc/pump.conf
# Set the timeout to 60 seconds
```

```
timeout 60
# Do not use a route through eth1 as the default
device eth1 {
    nogateway
}
```

The pump.conf file is a simple ASCII text file. Comments are indicated by the hash mark (#). Global parameters can be set for the entire system. In Listing 8.4, timeout is set as a global parameter. In this case, it tells pump to wait 60 seconds before timing out the DHCP negotiations. Parameters can also be set for individual devices. In Listing 8.4, the device statement identifies the second Ethernet interface. The curly braces that follow the device statement enclose directives that apply only to the second Ethernet interface eth1. The nogateway directive tells pump not to reset the client's default gateway, based on DHCP information that arrives through the eth1 interface. Perhaps eth0 is the primary interface, and you want to ensure that a default route learned from eth1 does not overwrite the default route that goes through eth0.

Both pump and dhcpcd have enough flexibility for any average system. If you have neither of these tools on your system, or you need to create a highly complex custom configuration, you can use dhclient, which is the DHCP client package provided with the dhcpd distribution.

Running *dhclient* Software

The philosophy of dhclient is very different from that of most DHCP clients, which assume that users run DHCP because they don't know how to (or don't want to) manually configure TCP/ IP. dhclient assumes that the people running the software are sophisticated users who can easily configure TCP/IP, and who want more than basic configuration from a DHCP client.

Many Linux systems do not include the dhclient software. If your Linux system doesn't have the client, download dhcpd from http://www.isc.org/. The dhcpd distribution includes the dhclient software.

After the client software is installed, it is run using the command dhclient. The command does not require any command–line arguments. When dhclient starts, it reads two files: the configuration file dhclient.conf and the lease file dhclient.leases. You create the configuration file; the lease file is created by dhclient, and is used to preserve information about IP address leases.

The *dhclient.leases* File

The dhclient.leases file contains a history of the address leases granted to the client. dhclient uses this information for address renewal and to speed the boot process. The leases are stored in a format that dhclient calls lease declarations, which look like the commands in the configuration file. The format of these entries is:

lease { *lease–description* [... *lease–description*] }

The lease declaration begins with the lease keyword, followed by one or more lease descriptions. If more than one command is used to describe a lease, they are enclosed in curly braces. The following commands are possible in a lease description:

> **bootp** Means that the lease was acquired from a BootP server rather than a DHCP server.

expire *date* Defines when dhclient must stop using the address lease if it has not renewed it. *date* in all of these commands is year, month, day, hour, minute, and second.

filename "*path*" Defines the pathname of the boot file for diskless clients.

fixed–address *address* Defines the address assigned by the lease. Despite the name fixed–address, this command is found in all lease statements, even those that assign dynamic addresses.

interface "*name*" Identifies the interface on which the lease is valid.

option *option–name value* Defines the value of a configuration option supplied by the server.

rebind *date* Defines when dhclient should connect to a new server if it has not renewed the address lease.

renew *date* Defines when dhclient should renew the address lease.

server–name "*hostname*" Defines the name of the boot server for diskless clients.

Using this information, you can read the dhclient.leases file to see what leases are active and when they expire. The lease declaration can also be used in the dhclient.conf file to manually define an address lease your system can fall back on if no server provides a valid lease. The next section looks at creating a dhclient.conf file.

The *dhclient.conf* Configuration File

The first—and most important—thing to know about the dhclient.conf file is that you can create an empty one, and dhclient will probably work fine. The dhclient.conf file allows you to change the default settings of the DHCP client. On most networks, this is unnecessary. However, the capability exists if you need to use it.

The manual page for dhclient.conf provides an example of a very complex client configuration. The sample in Listing 8.5 is a somewhat simplified version of that dhclient.conf file

Listing 8.5: A Sample *dhclient.conf* File

```
# Set the protocol timers
timeout 60;
retry 60;
select-timeout 5;
reboot 10;

# Define configuration parameters for eth0
interface "eth0" {
   send host-name "sparrow.foobirds.org";
   send dhcp-client-identifier 1:0:a0:24:ab:fb:9c;
   send dhcp-lease-time 28800;
   supersede domain-name "foobirds.org";
   prepend domain-name-servers 127.0.0.1;
   request subnet-mask, broadcast-address, routers,
           domain-name-servers;
   require subnet-mask, routers;
```

```
}

# Define a static address for emergencies
alias {
   interface "eth0";
   fixed-address 172.16.5.23;
   option subnet-mask 255.255.255.255;
}
```

Protocol Timers The first four lines of the sample file set values for the protocol timers used by dhclient:

> **timeout** Sets the maximum number of seconds that dhclient waits for a server to respond. If a server does not provide the client with an offer of an address lease within the timeout period, the client attempts to use any lease in the dhclient.leases file that has not expired, or any static address defined in the dhclient.conf file. The value set in the sample file, 60 seconds, is the default, so this command was not really required.
>
> **retry** Sets the number of minutes that dhclient waits before it tries again after it failed to get a response from a server. The default is five minutes. In the example, it is set to two minutes.
>
> **select–timeout** Sets the number of seconds the client waits after receiving information from a server to see if another server responds. More than one server may exist on a network. The servers may be configured differently, and may provide different levels of configuration information to the client, so sometimes it's worthwhile to "wait for a better offer." By default, the client does not wait. It takes the first offer. In the example, dhclient is configured to wait 2 seconds.
>
> **reboot** Sets the number of seconds the client tries to reacquire the address it used during the last boot. The system assumes that the client will reattach to the same subnet, and that the address it was last using is still available for it to use. Normally, these are very good assumptions, and making them speeds the reboot process. This didn't need to be specified in the configuration because it was set to 10 seconds, which is the default.

The *interface* Declaration Next comes the interface declaration. The interface declaration is used to define parameters that relate only to a specific interface. It is intended for use on computers that have more than one interface to allow you to configure each interface in a unique way. In the example, there is only one interface, so the command is not required. However, it does no harm, and it illustrates the use of the interface declaration. The curly braces enclose all of the parameters that relate to the specified interface.

***send* Statements** The first three items associated with the interface declaration are send statements, which tell dhclient to fill in values for the specified configuration options before sending the request packet to the server. Any of the options listed in the "dhcpd Configuration Options" section of this chapter, plus the option dhcp–lease–time, can be used in a send statement. Of course, only certain options make any sense when used in this way. Generally, the send statement is used to help the server identify the client. That's what's happening in the sample file with the host–name option and the dhcp–client–identifier option. The dhcp–lease–time option defines the length of the address lease requested by the client. If this option is not specified, dhclient defaults to requesting a two–hour lease. In the example, eight hours (28800 seconds) are requested.

The *supersede, prepend, request*, and *require Commands* A supersede command defines a value for a configuration option that is used instead of the value provided by the server. In the example, the domain−name option is defined with a value of foobirds.org. Regardless of the domain−name value provided by the DHCP server, the computer uses foobirds.org as a domain name.

The prepend command also affects the value provided for a configuration option. The value specified on the command line is inserted at the beginning of the list of values assigned to the option. The prepend command can be used only for configuration options that accept a list of values. In the example, the command is used to place the local host at the beginning of the list of name servers.

The request command defines the configuration options that dhclient requests from the server. These are requested in addition to an IP address. Any valid configuration option name can be listed. In the example, the subnet mask, the broadcast address, the default router, and the DNS servers are requested.

The require statement defines the configuration options that the client requires from a DHCP server. If the server does not provide this information, the client will not use any of the configuration information provided by the server. In the example, a subnet mask and a default router are required. As the curly brace indicates, this is the last command in the interface declaration.

The *alias* Declaration The configuration file ends with an alias declaration, which defines a static address that is used by the client if it does not receive a valid address lease from a server. The example file shows all of the elements of an alias declaration:

- An interface statement to define the interface being assigned an address
- A fixed−address option to define the address
- A subnet−mask configuration option to define the address mask

Other Configuration Commands This sample configuration is much more complex than any configuration you will ever create, but even a complex example such as the one in Listing 8.5 does not cover all of the configuration commands available for dhclient. Following are the other configuration commands you can use:

append {*option−list*} Adds values to the end of the list of values provided by the server. This statement can be used only for options that allow more than one value. The append command is essentially the opposite of the prepend command described earlier.

backoff−cutoff *time* Defines the maximum amount of time that the client is allowed to back off on configuration requests. The default is two minutes. A randomized exponential back−off algorithm prevents clients from all trying to configure themselves at the same time. This value prevents the client from backing off for an excessive amount of time.

default {*option−list*} Sets values for options that are used if the server does not supply the values. If the server provides a value, it is used; otherwise, the value defined here is used. For example, to make sure the client has a subnet mask, even if one is not provided by the server, you might enter the following:

```
default subnet-mask 255.255.255.0;
```

initial–interval *time* Sets the time interval that is multiplied by a random number between zero and one to determine the amount of time the client backs off between attempts to reach a server. The time between attempts is doubled on each failed attempt. If it exceeds the backoff–cutoff amount, it is set to that amount. The default time interval is 10 seconds.

media "*media setup*" [, ...] Defines media configuration parameters for network interfaces that aren't capable of sensing the media type unaided.

reject *address* DHCP offers from the server at the specified address are ignored.

script "*path*" Identifies the dhclient configuration script file. This is a script that configures the local system's network interface with the values learned from the DHCP server. For more information, see the manual page for dhclient–script.

Looking at this large configuration language and the complex example, you may be wondering why you would do this. After all, this is at least as challenging as statically configuring TCP/IP with ifconfig. The reason is primarily for mobility. A computer consultant may need to move a Linux laptop from network to network in the course of a business day. A tool such as dhclient eliminates the need to manually reconfigure the system every time it is attached to a new network.

In Sum

Configuring TCP/IP is complex for users who do not understand the technical aspects of networking. Centralizing the configuration on a server makes it simpler for users to connect to the network, and it makes it simpler for you to support your network.

The configuration service is the foundation service of a departmental network. It allows you to control the configuration of every system in your department. This simplifies the task of running a departmental network, and it allows you to point your clients to your other servers.

In the next chapter, the most fundamental of the other departmental services is configured. File sharing is the service that created the original demand for departmental networks.

Chapter 9: File Sharing

Overview

The ability to share information by sharing files is the fundamental service of a departmental network. Linux is a perfect system for this service because it provides a wide range of different file–sharing mechanisms that integrate Microsoft Windows clients, Unix clients, and other clients that are not compatible with either of these into a single, cohesive network.

Compared with proprietary servers that see the world in only one way, Linux servers provide increased flexibility for designing the right network. Linux does this by providing three distinct types of file sharing:

> **Mainframe technique** Allows clients to log in to the server and share files directly through the Linux filesystem. This model works with any client system that can emulate a terminal.
>
> **Unix network technique** Allows clients to share files across the network with the Network File System (NFS). NFS is the most popular file–sharing software on Unix networks.
>
> **Microsoft network technique** Allows clients to use the Server Message Block (SMB) protocol to share files across the network. SMB is the NetBIOS protocol used by Microsoft LanManager and Windows NT/2000 systems to provide file–sharing services to Microsoft Windows clients.

This chapter examines all three file–sharing techniques, beginning with the mainframe model that uses the basic capabilities of the Linux filesystem.

Linux Filesystem

telnet and ssh permit users to log in to the server and work together there on shared files. Using tools such as FTP and scp, files developed elsewhere can be placed on the server when users want to permit shared access to those files. As shown in Chapter 3, "Login Services," all that is required for this type of access is a user account for each user and the necessary daemons to provide the services. After a user successfully logs in to a Linux server, file sharing is controlled through the file permissions that exist in the Linux filesystem.

Linux File Permissions

When the user's account is created, every Linux user is assigned a user ID (UID) and a group ID (GID), which are used to identify the user for file access. Every file is also given a UID and GID. By default, these are the UID and GID of the person who creates it, though that can be changed. Permissions are granted based on matching the UIDs and GIDs of the file and the user as follows:

> **Owner permissions** The permissions granted to the user who has the same UID as the file. In addition to being called owner permissions, they are also called *user permissions*.

Group permissions The permissions granted to users who have the same GID as the file.

World permissions The permissions granted to all other users, those who have neither the UID nor the GID of the file. In addition to being called world permissions, they are also called *other permissions*.

Each of these groups can be granted any combination of three possible permissions:

Read permission The contents of the file may be examined.

Write permission The contents of the file may be modified.

Execute permission The program contained in the file may be executed.

Use the –l option with the ls command to view the ownership and permissions assigned to a file.

Listing 9.1: Examining File Permissions with *ls*

```
$ ls -l
total 1641
-rw-r--r--  1 craig  users    8255 May 17 14:09 fig2-1.gif
-rw-r--r--  1 craig  users    8206 May 17 14:10 fig2-2.gif
-rw-r--r--  1 craig  users   16328 May 16 22:04 fig3-2.gif
-rw-r--r--  1 craig  users    3832 May 16 22:13 fig4-1.gif
-rw-r--r--  1 craig  users   16741 May 16 22:18 fig4-2.gif
-rw-r--r--  1 craig  users   14350 May 16 22:24 fig4-4.gif
-rw-r--r--  1 craig  users   22737 May 16 22:27 fig4-5.gif
-rw-r--r--  1 craig  users   14316 May 16 22:34 fig5-1.gif
-rw-r--r--  1 craig  users   15739 May 16 22:35 fig5-2.gif
-rw-r--r--  1 craig  users   21528 May  1 20:46 fig8-1.gif
-rw-r--r--  1 craig  users   16479 May  1 21:18 fig8-2.gif
-rw-r--r--  1 craig  users   22295 May 17 11:43 fig8-4.gif
-rw-r--r--  1 craig  users   16482 Apr 24 19:50 fig9-3.gif
-rw-r--r--  1 craig  users   11756 Apr 24 19:54 fig9-4.gif
```

Each line in the long format directory listing begins with the file permissions. The first 10 characters are the same for every file in Listing 9.1: –rw–r––r––. The very first character indicates whether this is a directory (d), a link (l), or a file (–). In the example, all of the entries are files.

The next nine characters are divided into three groups of three to define the permissions for the owner of the file, for the members of the group to which this file is assigned, and for all other users. An r in the permission field indicates read permission, a w indicates write, and an x indicates execute.

In Listing 9.1, the owner is granted read and write permissions (rw–), and everyone else— members of the group as well as other users of the system—are granted only read access (r––).

The permissions can be viewed as three 3–bit numbers. r is 4 (binary 100), w is 2 (binary 010), and x is 1 (binary 001). Thus, the permission granted to the owner in Listing 9.1 is 6 (rw–), and the permissions granted to the group and to the world are 4 (r––) for a file permission setting of 644.

Changing File Permissions

Use the chmod (Change Mode) command to change the permissions for a file. Permission can be defined on the chmod command line in either numeric or symbolic formats. For example:

```
$ ls -l trace.txt
-rw-r--r--  1 craig  craig   1349 May  3  2000 trace.txt
$ chmod g+w,o-r trace.txt
$ ls -l trace.txt
-rw-rw----  1 craig  craig   1349 May  3  2000 trace.txt
```

The first ls command shows the current permissions assigned to the file trace.txt, which are owner read/write, group read, and world read. The chmod command uses the symbolic format for defining permissions. g+w tells chmod to use the current group permissions and add write permission, and o–r tells chmod to use the current world (or other) permissions and subtract read permission. The second ls command shows the effect that this chmod command has on the trace.txt file permissions.

The symbolic chmod format has three fields. The first defines whether the permission is being set for the owner, group, world, or all three. u sets permission for the owner, which is also called the *user*. g sets permission for the group. o sets permission for the world, which is also called *other*. And a sets permissions for the owner, group, and world.

The second field defines how the permissions are applied. Permission can be added to existing permission by placing a + in the second field, they can be subtracted from existing permissions by using a –, or they can replace the existing permissions by using an = in the second field.

The third field defines the specific permissions. r, w, and x are read, write, and execute, respectively. s is used for SetUID and SetGID permissions. t sets the sticky bit. (See the sidebar "Hidden Bits" for more information about these permissions.) Additionally, the permissions that are already defined for the owner, group, or world can be assigned to one of the other groupings by using u, g, or o in the permission field. For example, g=u would set the group permissions to exactly the same values that were previously defined for the owner permissions.

Of course, chmod permissions do not have to be defined symbolically. Numeric permission can also be used. For example:

```
$ chmod 777 test.pl
$ ls -l test.pl
-rwxrwxrwx  1 craig  users  16513 May 18 14:22 test.pl
```

In this example, the permission is changed to 777, which grants read, write, and execute permissions to the owner, to the group, and to the world. The ls –l command illustrates what this full array of permissions looks like in a directory listing. It is unlikely, however, that you will want to grant such liberal permissions. It is more likely that you will want to offer less than the 644 you saw earlier, particularly if you don't want everyone who logs in to the system to be able to read your private files. To prevent those outside your group from reading your report before it is released, you might use the following setting:

```
$ chmod 640 report.txt
$ ls -l report.txt
-rw-r-----  1 craig  users  16513 May 18 14:22 report.txt
```

This setting permits the owner to read and write the file and the other members of the group to read the file, but it blocks general users from accessing the file at all. This is better, but it is still not enough. The problem is that the group to which this file is assigned is too broad.

Hidden Bits

So far, this discussion of file permissions is limited to user file permissions. Read, write, and execute are types of permission granted to various classes of users. There are also some permissions that are used to grant special privileges to executable files. These permissions are the following:

- Sticky bit, which permits the program to remain in memory after execution. The Sticky bit is an artifact of an earlier age; programs don't really need to stay in memory anymore. A more common use for the Sticky bit is to use it with directories instead of files. When used with a directory, users may delete only files for which they have specific write permission, even if they have directory write permission.
- SetGID, which permits the program to set the group ID it runs under on execution. When used on a directory, SetGID means that all files created in the directory belong to the directory's group by default.
- SetUID, which permits the program to set the user ID it runs under on execution.

These three permissions create another group of three permission bits. The Sticky bit is set by the value 1 (binary 001), the SetGID permission is set by value 2 (binary 010), and the SetUID permission is set by value 4 (binary 100). All are set by placing a fourth digit at the beginning of the file permission value. Therefore, to grant a file SetUID permission; read, write, and execute permissions for the owner and the group; and execute permission for the world, you would use the value 4771 with the chmod command. 4 sets the SetUID permission, the first 7 sets the owner permission, the second 7 sets the group permission, and the 1 sets the world permission.

At first glance, these permissions don't appear to have a place in the three–character display (rwx) shown in Listing 9.1. But that's not really true. ls shows all of these permissions as alternate values in the execution bit:

- If the Sticky bit is set, an uppercase letter "T" appears in the world execute permission field.
- If both world execute and the Sticky bit are set, a lowercase letter "t" appears in the world execute permission field.
- If SetGID is set, an uppercase letter "S" appears in the group execute permission field.
- If both group execute and the SetGID bit are set, a lowercase letter "s" appears in the group execute permission field.
- If SetUID is set, an uppercase letter "S" appears in the owner execute permission field.
- If both owner execute and the SetUID bit are set, a lowercase letter "s" appears in the owner execute permission field.

Be careful when granting programs SetUID and SetGID permissions. If these programs are owned by the root user, they can have a great deal of power over the system. If the programs are incorrectly written, they can be exploited by an intruder and compromise your entire system.

The *chgrp* Command

The ls –l command lists the username of the file owner and the group name assigned to the file. In all of the examples shown previously, the user is craig, and the group is users. By default, the file is assigned the owner's primary GID, which is the GID assigned to the user in the /etc/passwd file. This might not be what you want, particularly if access to the files should be limited to a group that logs in to the system to jointly work on the file. Use the chgrp command to change the group of a file:

```
$ chgrp rnd report.txt
$ ls -l report.txt
-rw-r----- 1 craig  rnd 16513 May 18 14:22 report.txt
```

In this example, the group of the report.txt file is changed to rnd. Now, craig has read and write permissions for this file, and anyone in the group rnd has read permission. All other users have no access at all.

> **Note** craig, who is the owner of the file, must be a member of the rnd group to change the file to that group. Unless you're the root user, you cannot change a file to a group of which you're not a member.

This is a good example of how files are shared using the "mainframe model." Anyone in the rnd group can copy the report.txt file to their home directory where they can modify the copy by adding their comments and changes. The person in charge of the report can then look at the comments and changes using a command such as diff, and decide what should be included in the final copy. Only the person in charge of the report can actually write to the master copy.

Sharing files by logging in to the server can work with any client system. It doesn't even require a client computer; a terminal will work just fine. But most users have powerful desktop computers that have the software tools that they like best. A better way to share files lets users work with the files on their desktop systems with the applications they like. Both Unix and Microsoft Windows offer such a service. For Unix systems, Network File System is the most popular service that provides this type of file sharing.

Understanding NFS

The *Network File System (NFS)*, originally developed by Sun Microsystems, allows directories and files to be shared across a network. Through NFS, users and programs access files located on remote systems as if they were local files.

NFS is a client/server system. The client uses the remote directories as if they were part of its local filesystem; the server makes the directories available for use. Attaching a remote directory to the local filesystem is called *mounting* a directory. Offering to share a direc tory is called *exporting* a directory.

NFS is a remote procedure call (RPC) protocol that runs on top of UDP and IP. A remote procedure call is simply a system call that is processed by a remote server. When a program makes an I/O call for an NFS file, the call is intercepted by the NFS filesystem, and is sent over the network to the remote server for processing.

The daemons that process the NFS requests on the server are not assigned standard UDP port

numbers. Instead, they are dynamically assigned port numbers by the RPC portmapper. On some systems, the portmapper program is named rpc.portmap. On Red Hat systems, the program is named portmap, and is started by the script /etc/rc.d/init.d/portmap.

Use the rpcinfo command to view the port numbers that portmapper has assigned to various RPC services.

Listing 9.2: Displaying RPC Ports

```
$ rpcinfo -p
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp  32768  status
    100024    1   tcp  32768  status
    100011    1   udp    687  rquotad
    100011    2   udp    687  rquotad
    100011    1   tcp    690  rquotad
    100011    2   tcp    690  rquotad
    100005    1   udp  32769  mountd
    100005    1   tcp  32771  mountd
    100005    2   udp  32769  mountd
    100005    2   tcp  32771  mountd
    100005    3   udp  32769  mountd
    100005    3   tcp  32771  mountd
    100003    2   udp   2049  nfs
    100003    3   udp   2049  nfs
    100021    1   udp  32770  nlockmgr
    100021    3   udp  32770  nlockmgr
    100021    4   udp  32770  nlockmgr
```

Port number 111 (portmapper) is the portmapper's own port number, which is a well–known port number assigned in the /etc/services file. (On some systems, the name associated with this port may be rpcbind or sunrpc.) Remote systems can contact the portmapper because it is on a standard port. From the portmapper, the remote systems learn the ports used by other RPC services. All of the other port numbers in this listing from our Red Hat 7.2 Linux server are assigned to NFS daemons. If other RPC services, such as NIS, were running on this server, portmapper would also list those services in response to the rpcinfo command. In this case, only NFS is running. The various NFS daemons, in the order that they appear in the listing, are as follows:

status The status monitor reports crashes, and reboots to the lock manager so that file locks can be properly reset if an NFS client reboots without gracefully terminating its NFS connection. rpc.statd monitors both TCP and UDP traffic.

rquotad The remote quota server, rpc.rquotad, enforces filesystem quotas for NFS mounted filesystems. Filesystem quotas control the amount of disk storage an individual user can consume. This daemon extends that feature to NFS users.

mountd The mount daemon processes the client's filesystem mount requests. rpc.mountd is the program that checks whether or not a filesystem is being exported, and whether or not the client making the request is allowed to mount the requested filesystem.

nfs The nfsd program handles the user–level interface to the NFS kernel module (nfsd.o). The NFS file I/O is handled in the kernel module.

nlockmgr The NFS lock manager (lockd) handles file–lock requests from clients. File locking is used to prevent file corruption when it is possible that multiple sources may attempt to write a file at the same time. Read–only files do not require file locking.

These daemons are part of the kernel NFS implementation that is included with the Linux 2.4 kernel. If you have an older system with a user–level implementation of NFS, you will see a different list of daemons, and future versions of Linux may have a somewhat different list. Regardless of the detail differences, an NFS daemon to handle file I/O and a mount daemon to handle mount requests will certainly be included in your implementation of NFS.

Installing NFS

Listing 9.2 shows that the Network File System includes several different daemons and services to perform client and server functions. Additionally, the Red Hat 7.2 distribution has multiple startup scripts in the /etc/rc.d/init.d directory that relate to NFS:

nfs This script starts most of the NFS daemons. It also processes the exports file, and clears the lock file. The exports file and the exportfs command that is used to process it are covered later in this chapter.

nfslock This script starts the NFS lock management software lockd and rpc.statd.

netfs This script mounts the NFS filesystems listed in the /etc/fstab file. Red Hat also uses this script to mount SMB filesystems. (SMB, the mount command, and fstab are covered later in this chapter.) Some other Linux systems don't use a separate script for this purpose; instead, they rely on the mount –a command to mount everything in the fstab file. The Red Hat approach mounts filesystems that require the network separately from filesystems that don't need a functioning network connection to ensure that network problems do not delay the mounting of local filesystems.

amd This script starts the automounter daemon (amd) that automatically mounts filesystems when the files they contain are accessed. (For a definitive, book–length treatment of amd, see *Linux NFS and Automounter Administration*, by Erez Zadok, Sybex, 2001.)

autofs This script starts the automounter filesystem (automount) that automatically mounts files when they are needed, and dismounts them when they are not in use. automount and amd are two different implementations of an automounter.

NFS is included in most Linux distributions. To install it on your system, select the necessary software component during the initial installation. If your Linux system is already running, but the NFS software is not yet installed, use a package manager, such as rpm or gnorpm, to install the software. Figure 9.1 uses the gnorpm software to check the NFS software that is installed on our sample Red Hat system.

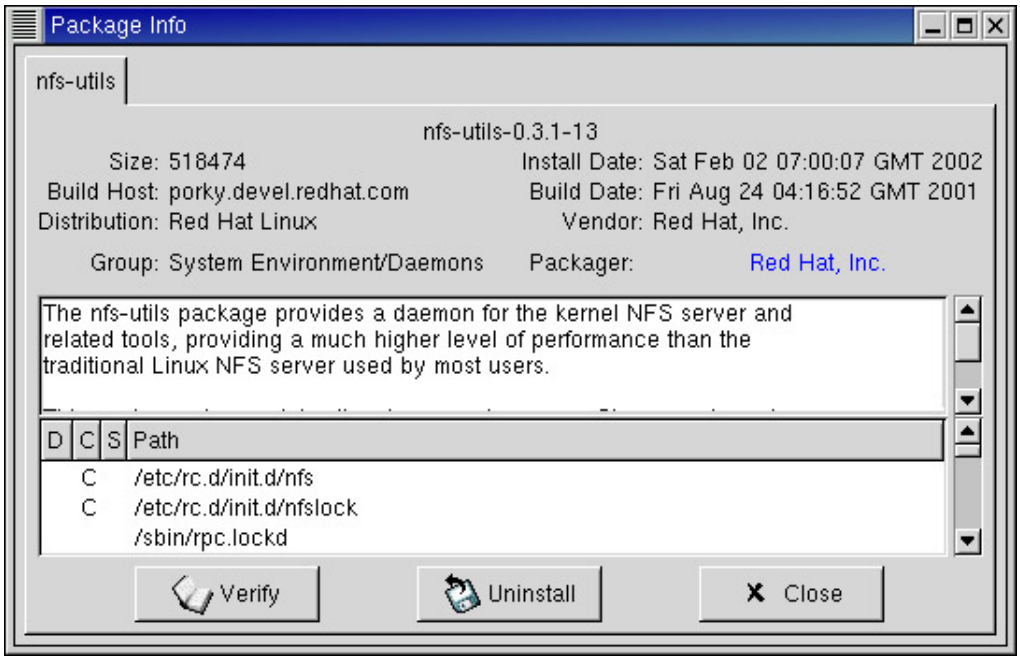


Figure 9.1: The Red Hat NFS RPM

The RPM described in Figure 9.1 provides user–space tools for the kernel NFS server software. Because the NFS server uses kernel–level code, it must be configured in the kernel. (The CONFIG_NFSD kernel option enables the kernel–level NFS server.) Additionally, all Linux systems that use NFS should have the CONFIG_NFS_FS kernel option selected. (See Chapter 13, "Troubleshooting," for a discussion of kernel configuration.)

Use a tool such as tksysv or chkconfig to enable the NFS startup scripts, so that NFS will restart the next time the system reboots. Of course, you don't have to reboot to start these services. After the software is installed, and you have ensured that the scripts are enabled, enter service nfs start to manually start NFS:

```
[root]# service nfs start
Starting NFS services:  [  OK  ]
Starting NFS quotas: [  OK  ]
Starting NFS mountd: [  OK  ]
Starting NFS daemon: [  OK  ]
```

Check the process status to see if the necessary daemons are running:

```
[root]# ps -Cnfsd
  PID TTY          TIME CMD
 1368 pts/0    00:00:00 nfsd
 1371 pts/0    00:00:00 nfsd
 1372 pts/0    00:00:00 nfsd
 1373 pts/0    00:00:00 nfsd
 1374 pts/0    00:00:00 nfsd
 1375 pts/0    00:00:00 nfsd
 1376 pts/0    00:00:00 nfsd
 1377 pts/0    00:00:00 nfsd
[root]# ps -Crpc.mountd
  PID TTY          TIME CMD
 1363 ?        00:00:00 rpc.mountd
```

In this example, eight copies of the user–level nfsd server process are running to support user connections. By default, the rpc.nfsd command starts only one process. Starting additional processes is a command–line option. In the Red Hat nfs script, this command–line option is set by

the RPCNFSDCOUNT script variable. To change the number of nfsd processes running, change the number assigned to that variable. Normally, eight nfsd processes are enough to provide very good service. If your server is extremely busy, you can try increasing this number to improve performance.

If everything is installed, and the startup scripts have been run, but the daemons are still not running, it may be because you haven't yet configured the server. Some startup scripts check to see if the /etc/exports configuration file exists before starting the daemons. Your next task is to create this file.

Configuring an NFS Server

The /etc/exports file is the NFS server configuration file. It controls which files and directories are exported, which hosts can access them, and what kind of access is allowed. The general format of entries in the /etc/exports file is

directory [*host*(*option*)]...

The *directory* variable is the full pathname of the directory or file being exported. If the directory is not followed by a host or an option, all hosts are granted read/write access to the directory.

The *host* variable is the name of the client granted access to the exported directory. If no host value is specified, the directory is exported to everyone. Valid host values are the following:

- Individual host names such as parrot.foobirds.org.
- Domain wildcards such as *foobirds.org for every host in the foobirds.org domain.
- IP address/address mask pairs such as 172.16.5.0/255.255.255.0 for every host with an address that begins with 172.16.5.
- Net groups such as @group1. A *net group* is a name assigned to a group of individual hosts in the /etc/netgroup file. They are primarily used on systems that run NIS. (See the netgroup manual page for more details.)

The *option* variable defines the type of access being granted. If the *option* is specified without a hostname, the access is granted to all clients. Otherwise, the access is granted only to the host that is named. The two most common options are

ro Specifies that clients may read only from the directory. Writing to the directory is not permitted.

rw Grants full read and write access to the directory. Read/write access is the default permission.

In addition to these two common options, there are several options that relate to UIDs and GIDs. NFS uses UIDs and GIDs to control file access in the same way that they are used to control access to local files. However, the fact that NFS must deal with the UIDs and GIDs assigned on several different systems means that coordination problems can be encountered. The options that help you work around these coordination problems are described in the upcoming "Mapping User IDs and Group IDs" section of this chapter.

A realistic sample /etc/exports file might contain the entries shown in Listing 9.3.

Listing 9.3: A Sample */etc/exports* File

```
/usr                    172.16.5.0/255.255.255.0(ro)
/home                   172.16.5.0/255.255.255.0(rw)
/usr/local/man          flicker(rw) parrot(rw)
/usr/local/doc          flicker(rw) parrot(rw)
/usr/local/bin              hawk(rw)
/home/sales             *.sales.foobirds.org(rw)
```

The first entry in this file grants read–only access to the /usr directory to every client on network 172.16.5.0. In the example, the network is defined with an IP address and an address mask. Assuming that 172.16.5.0 is the local subnet, this entry grants access to everyone on the local network without also granting access to everyone in the local domain, or without trying to list all of the hosts on the local network. The /usr directory contains documentation and executables that could be of interest to any Linux client. Read permission is all that is required to access those useful files.

The second line in the example grants read and write access to the /home directory. Again, the access is given to every host on the local subnet. Perhaps the /home directory is being exported to give users NFS access to their home directories on the server. To make full use of their directories, the users require read and write permissions.

The next three lines all grant individual hosts read and write access to specific directories within the /usr directory. These entries do not affect the first line of the file. The /usr directory is still exported as read–only to all local clients. Older versions of NFS that ran under Unix did not let you export a subdirectory of a directory you already exported. Linux does, however, and it can be very useful. These additional entries were added so that the people who maintain the documentation in /usr/local/doc and /usr/local/man can modify the documentation directly from their desktop systems, and so the people who maintain the executables in /usr/local/bin can do it from their desktops.

The last line in the file exports the /home/sales directory to every host in the sales.foobirds.org subdomain. In this case, the /home/sales directory is probably used by the sales division to share files. As this shows, it is possible for the server to share directories with computers in other domains or networks.

Even though specific hosts have been granted read/write access to some of these directories, the access granted to individual users of those systems is controlled by standard Linux user, group, and world file permissions based on the user's UID and GID. Essentially, NFS trusts that a remote host has authenticated its users and assigned them valid UIDs and GIDs, which is sometimes called the *trusted host* security model. Exporting files grants the client system's users the same access to the files they would have if they directly logged in to the server.

For example, assume that the server exporting these files is wren. Further, assume that user craig has accounts on both wren and eagle, and that both systems assign him UID 501 and GID 206. Everything works fine! But what happens if hawk has a user named david, and assigns him UID 501 and GID 206? The david account now has the same access to Craig's files as the craig account. That might not be what you intended. Linux provides tools to ease this problem.

Mapping User IDs and Group IDs

User IDs and group IDs are as fundamental to NFS as they are to any other part of the Linux filesystem. But unlike the UID and GID you assign when creating new user accounts, you may not

have any control over the UIDs and GIDs assigned by the clients of your NFS server. NFS provides several tools to help you deal with the possible problems that arise because of this.

One of the most obvious problems with a trusted host security model is dealing with the root account. It is very unlikely that you want people with root access to your clients to also have root access to your server. By default, NFS prevents this with the root_squash setting, which maps requests that contain the root UID and GID to the nobody UID and GID. Thus, if someone is logged in to a client as root, he is only granted world permissions on the server. You can undo this with the no_root_squash setting, but do so with caution. The no_root_squash setting opens more holes for intruders to exploit, which is never a good thing.

Use all_squash to map every user of a client system to the user nobody. For example, the following entry exports the /pub directory to every client:

```
/pub            (ro,all_squash)
```

It grants read–only access to the directory to every client, and limits every user of those clients to the world permissions granted to nobody. Therefore, the only files the users can read are those that have world read permission.

It is also possible to map every user from a client to a specific user ID or group ID. The anonuid and anongid options provide this capability. These options are most useful when the client has only one user, and the client does not assign that user a UID or GID.

The perfect example of this is a Microsoft Windows PC running NFS. PCs generally have only one user, and they don't use UIDs or GIDs. To map the user of a PC to a valid user ID and group ID, enter a line such as this in the /etc/exports file:

```
/home/kristin  robin(all_squash,anonuid=1001,anongid=1001)
```

Here, the hostname of Kristin's PC is robin. The entry grants that client read/write access to the directory /home/kristin. The all_squash option maps every request from that client to a specific user ID; but this time, instead of nobody, it maps to the UID and the GID defined by the anonuid and anongid options. Of course, for this to work correctly, 1001:1001 should be the UID and GID pair assigned to kristin in the /etc/passwd file of the server.

Coordination among the system administrators and perhaps some minimal UID/GID mapping are generally sufficient to handle the mapping problem. But the key is coordination and a clear plan about which servers will share files and why those files need to be shared. The following sidebar, "Coordinating UIDs and GIDs," describes the questions asked when one organization decided to create an enterprise–wide NFS service.

Coordinating UIDs and GIDs

When files are shared across a network, the systems need to identify users in a consistent and coordinated manner. Resolving user identity is an important security issue in a file–sharing network. The bigger the network, the bigger the potential problem. NFS includes a few tools for mapping UIDs and GIDs. But planning and coordination are the primary tools for avoiding this problem.

An organization with 5000 employees, composed of eight different operating units that had 50 different subnets, decided to create a unified NFS system that spanned the entire enterprise. For this system to permit true file sharing required that UIDs and GIDs be coordinated across the enterprise. However, the underlying network, subnets, and servers upon which this new service was

to be built already existed, and many of the subnets had their own network administrators and ran their own NFS services. This raised questions about the feasibility and even the desirability of the new service.

First, is file sharing across an enterprise really a good idea? True file sharing is when people cooperate to produce the end product. You don't really want 5000 people contributing to your report, so true file sharing across a very large organization is not usually what is wanted.

What most enterprise networks want is file dissemination. They want to make a finished product available to everyone in the organization. The Web is a good vehicle for this when the product is information, and NFS is a good choice when the product is an executable program.

Second, do you really need to coordinate UIDs and GIDs if what you want to do is disseminate information? Not really. The user nobody has world permission. Items being disseminated to the entire enterprise can be given world read and execute permissions.

Well, if enterprise file sharing wasn't really what the organization wanted, did anything good come from this project? Yes! A plan to assign UIDs and GIDs in a sensible coordinated manner. The central services held all of the UIDs and GIDs above 20,000 in order to assign them to every employee. Every existing employee was assigned a UID and a GID, and every new employee is assigned one when they come on board. Further, each operating unit was given 1000 UIDs and GIDs to use as they saw fit.

What began as a way to centralize services became a benefit to all of the independent subnet administrators. When a user from another organization is added to a subnet's NFS server, there is no worry about mapping or conflicts because a central authority coordinates UIDs and GIDs for the whole organization. In the end, enterprise–wide file sharing was not as important as enterprise–wide cooperation and coordination.

The *exportfs* Command

After defining the directories to export in the /etc/exports file, run the exportfs command to process the exports file and to build /var/lib/nfs/xtab. The xtab file contains information about the currently exported directories, and it is the file that mountd reads when processing client mount requests. To process all of the entries in the /etc/exports file, run exportfs with the –a command–line option:

```
# exportfs -a
```

This command builds a completely new xtab file based on the contents of the /etc/exports file.

It is also possible to use the exportfs command to update an existing xtab file to reflect changes to the /etc/exports file. If you edit the /etc/exports file and want the changes to take effect without replacing the current xtab, use the –r argument:

```
# exportfs -r
```

The –r option causes exportfs to synchronize the contents of the exports file and the xtab file. Items that have been added to the exports file are added to the xtab file, and items that have been deleted are removed from xtab.

You can even use the exportfs command to export a directory that is not listed in the /etc/exports file. For example, assume that you wanted to temporarily export the /usr/local directory to the client falcon with read/write permission. You could enter this command:

```
# exportfs falcon:/usr/local -o rw
```

Notice that the –o option is used to specify export options for the exported filesystem. In this example, the option is read/write (rw).

After the client has completed its work with the temporarily exported filesystem, the directory can be removed from the export list with the –u option. This command would end the export and prevent falcon from mounting the /usr/local directory:

```
# exportfs -u falcon:/usr/local
```

In fact, the –u option can be combined with the –a option to completely shut down all exports without terminating the NFS daemons:

```
# exportfs -ua
```

Despite all of its features, exportfs generally is not used to export individual directories. Usually, it is used with either the –a or the –r option to process all of the entries found in the /etc/exports file. For example, the Red Hat nfs script runs exportfs –r before starting the daemons.

After the daemons are running and the exports file has been processed by exportfs, the clients can mount and use the filesystems offered by your server. The next section looks at how a Linux system is configured as an NFS client.

Configuring an NFS Client

To configure an NFS client, you need to know the hostname of the NFS server and the directories it exports. The name of the server is usually very well advertised—no one creates a server unless they want to have clients. The network administrator tells users which systems are NFS servers.

The Linux showmount command lists the directories that a server exports and the clients permitted to mount those directories. For example, a showmount –-exports query to wren produces the output shown in Listing 9.4.

Listing 9.4: The *showmount* Command

```
$ showmount --exports wren
Export list for wren:
/home/sales     *.sales.foobirds.org
/usr            172.16.5.0/255.255.255.0
/home           172.16.5.0/255.255.255.0
/usr/local/doc  flicker.foobirds.org,parrot.foobirds.org
/usr/local/man  flicker.foobirds.org,parrot.foobirds.org
/usr/local/bin      hawk.foobirds.org
```

The showmount command lists the NFS directories exported by wren, and which clients are allowed to mount those directories. You can mount any of the directories offered by wren if you are a user on one of the approved clients.

The *mount* Command

Before using an NFS directory, attach it to the local filesystem with the mount command. The mount command can be as simple or as complex as it needs to be to get the job done.

At its simplest, mount identifies the remote filesystem to access and the local directory through which it will be accessed. The remote filesystem is identified by the server name, paired with all or part of a directory exported by the server. The local directory is just that—the name of an empty directory created to mount the remote NFS directory. The local directory is called the *mount point*. Putting this all together, you could mount the directories exported by wren with the mount commands shown in Listing 9.5.

Listing 9.5: Sample Mount Commands

```
# mount wren:/usr/local/bin /usr/local/bin
# mount wren:/usr/local/man /usr/local/man
# mount wren:/usr/local/doc /usr/local/doc
```

The Listing 9.5 examples assume that empty /usr/local/bin, /usr/local/man, and /usr/local/doc directories existed on the client before the mount commands were issued. It wouldn't make sense to mount a remote directory full of manual pages over an existing directory unless that directory were empty. The purpose for creating central repositories for man pages and documentation is to save storage on client systems and simplify management. You can do that only if the directories on the client are actually empty.

A simple mount command works under most circumstances, but when needed, options can be added to the mount command line with the –o argument. Table 9.1 lists the mount command options that apply to all types of filesystems.

Table 9.1: Linux mount Command Options

Option	Purpose
async	Use asynchronous file I/O.
atime	Update the inode access time for every access.
auto	Mount when –a option is used.
defaults	Set rw, suid, dev, exec, auto, nouser, and async.
dev	Allow character devices, and block special devices on the filesystem.
exec	Permit execution of files from the filesystem.
_netdev	Indicates a filesystem that depends on the network.
noatime	Don't update inode access time.
noauto	Don't mount with the –a option.
nodev	Don't allow character devices, and block special devices on the filesystem.
noexec	Don't allow execution of files from the filesystem.
nosuid	Don't allow programs stored on the filesystem to run setuid or setgid.
nouser	Only root can mount the filesystem.
remount	Remount a mounted filesystem with new options.
ro	Mount the filesystem read–only.
rw	Mount the filesystem read/write.

suid	Allow programs to run setuid or setgid.
sync	Use synchronous filesystem I/O.
user	Permit ordinary users to mount the filesystem.

Note Despite the length of this list, you will see even more NFS mount options in the next section.

Assume that you want to mount the /usr/local/bin directory, but for security reasons you don't want to allow any of the programs stored there to run with setuid or setgid permission. You could enter the following mount command:

```
# mount -o nosuid wren:/usr/local/bin /usr/local/bin
```

The *umount* Command

The opposite of the mount command is the umount command, which is used to remove a mounted directory from the local filesystem. A filesystem can be dismounted using either the remote filesystem name or the local mount point directory on the umount command line, so to dismount the /usr/local/bin directory, enter either the remote name

```
# umount wren:/usr/local.bin
```

or the local name

```
#umount /usr/local/bin
```

There are a few options associated with the umount command that are of particular interest. The –a and –t options are used in the same way that they are used with the mount command. The –a option dismounts every filesystem listed in the /etc/mtab file. In other words, it dismounts all currently mounted filesystems. The –t option, when combined with the –a option, dismounts all filesystems of the specified type. The –f option has particular relevance to NFS because it forces the dismount, even if the remote NFS server is unresponsive.

Using *fstab* to Mount NFS Directories

A mount command with the –a flag set causes Linux to mount all filesystems listed in /etc/fstab. Linux systems often include a mount –a command in the startup. Adding the –t nfs argument to the mount –a command limits the mount to all filesystems in fstab that have a filesystem type of NFS. The Red Hat netfs script uses the –t nfs argument to remount the NFS filesystems after a system boot.

The filesystem table, /etc/fstab, defines the devices, partitions, and remote filesystems that make up a Linux computer's complete filesystem. Each line in the table describes an individual piece of the filesystem. These pieces can be hard disk partitions, devices such as the floppy drive or the CD–ROM drive, or a filesystem from a remote NFS server. The file includes entries for the root partition, the swap partition, and even the pseudo filesystems, such as /proc. A sample fstab file from a dual–booting desktop system running Red Hat Linux 7.2 is shown in Listing 9.6.

Listing 9.6: A Sample *fstab* File

```
$ cat /etc/fstab
LABEL=/        /              ext3    defaults        1 1
LABEL=/home    /home          ext3    defaults        1 2
/dev/fd0       /mnt/floppy    auto    noauto,owner    0 0
```

```
LABEL=/var    /var           ext3    defaults        1 2
/dev/hda1     /win           vfat    defaults        0 0
none                    /proc         proc     defaults               0 0
none                    /dev/shm      tmpfs    defaults               0 0
none                    /dev/pts      devpts   gid=5,mode=620         0 0
/dev/hda2               swap          swap     defaults               0 0
/dev/cdrom              /mnt/cdrom    iso9660  noauto,owner,kudzu,ro  0 0
crow:/export/home/craig /home/craig   nfs      rw                     0 0
```

Each line in the fstab file is composed of six fields. The first field is the filesystem name. It is the keyword none for pseudo filesystems, such as /proc. For local devices, the filesystem name field contains the name of the local device. It must be the name of a "block special device." On Linux systems, mknod can create two types of devices: character devices, which provide data one character at a time, and block devices, which provide chunks of data. Use the ls –l command to find out which type a device is:

```
$ ls -l /dev/hda1
brw-rw----  1 root   disk    3,  1 Aug 30  2001 /dev/hda1
$ ls -l /dev/ttyS0
crw-rw----  1 root   uucp    4, 64 Aug 30  2001 /dev/ttyS0
```

The first character in the output of the ls command shows the device type. If it is a c, the device is a character device. If it is a b, the device is a block device. All of the local devices in the fstab file are block devices, but they are not all referred to by name. The filesystem name field can contain a label if that label is mapped to a local block special device. Listing 9.6 contains three entries with label values in the filesystem name field:

```
LABEL=/       /              ext3    defaults        1 1
LABEL=/home   /home          ext3    defaults        1 2
LABEL=/var    /var           ext3    defaults        1 2
```

Use the e2label command to see the label associated with a device. For example:

```
$ e2label /dev/hda3
/
$ e2label /dev/hdd1
/home
$ e2label /dev/hdd2
/var
```

The filesystem name field in an entry for an NFS filesystem is formatted in exactly the same way as the remote filesystem name on a mount command. It contains the hostname of the NFS server, separated from the filesystem pathname by a colon. For example, in crow:/export/home/craig, crow is the hostname of the NFS server, and /export/home/craig is the pathname of the remote filesystem.

The second field in each fstab entry is the mount point for the filesystem. Most of the mount points in Listing 9.6 are self–explanatory. Some are the mount points you created when partitioning the disk. Some are mount points for hardware devices, such as the floppy and the CD–ROM. A few are the mount points for pseudo filesystems. And in Listing 9.6, one is an empty directory that we created to act as a mount point for an NFS filesystem. All of these appear as directories within the root directory hierarchy. The clear exception is the swap space; it does not have a mount point, and no swap subdirectory appears below the root directory. On Linux systems, the keyword swap appears in this field for the swap partition entry.

The third field defines the filesystem type. Linux 2.4 can support more than 30 different filesystem types. Not all of them will be used on your system, and not all will have support compiled into your kernel. In Listing 9.6, eight different filesystem–type keywords are specified:

- ext3 is the most important—it is a native Linux journaling filesystem. Most Linux systems use either ext2 or ext3 as their primary filesystem.
- swap is a special format used for the swap file.
- iso9660 is the International Standards Organization (ISO) format for CD–ROMs.
- auto is not really a filesystem type. It tells mount to probe the device to determine the filesystem type. This is useful for the floppy device because Linux can read different types of floppies.
- vfat is the Microsoft FAT format with support for long names. This is a dual–booting desktop system, so it has Windows installed in one of the partitions. Linux can read these Windows files.
- The three remaining types (tmpfs, devpts, and proc) are pseudo filesystems. Unless you want to watch virtual terminals pop up in the /dev/pts directory, the only one of them that contains information of interest to a system administrator is the /proc filesystem. The /proc filesystem provides an interface into the kernel data structures that provide information about running processes.

The fourth field is a comma–separated list of options. The defaults option identifies filesystems that can do the following:

- Be mounted at boot time by the mount –a command
- Be mounted as read and write filesystems
- Allow SetUID and SetGID processes
- Support both character and block devices

In other words, they are typical Linux filesystems.

The noauto option indicates filesystems that should not be mounted by the mount –a command. In Listing 9.6, these are the floppy and CD–ROM devices. The ro option on the CD–ROM indicates that it is a read–only device, and the owner option indicates that it can be mounted and unmounted only by the device owner. The gid and mode options define the GID and file permission used to create files in the /dev/pts directory. There are many more options that can appear in the fstab file, some of which are listed in Table 9.1, and more of which can be found in Table 9.2.

Table 9.2: More mount Options

Option	Function
acdirmax=*n*	Sets the maximum cache time for directory attributes. Defaults to 60 seconds.
acdirmin=*n*	Sets the minimum cache time for directory attributes. Defaults to 30 seconds.
acregmax=*n*	Sets the maximum cache time for file attributes. Defaults to 60 seconds.
acregmin=*n*	Sets the minimum cache time for file attributes. Defaults to 3 seconds.
actimeo=*n*	Sets all cache times to the same value.
bg	Does retries in background mode.
fg	Does retries in foreground mode.
hard	Retries indefinitely until the server responds.

intr	Allows a keyboard interrupt to kill a process.
mounthost=*name*	Sets the name of the server running mountd.
mountport=*n*	Sets the port number of mountd.
mountprog=*n*	Uses an alternate RPC program number for mountd on the remote server.
mountvers=*n*	Uses an alternate RPC version number for the mountd on the remote server.
namlen=*n*	Sets the maximum length of a filename for the remote filesystem. Defaults to 255 bytes.
nfsprog=*n*	Uses an alternate RPC program number for nfsd on the remote server.
nfsvers=*n*	Uses an alternate RPC version number for nfsd on the remote server.
noac	Disables all caching.
nocto	Does not retrieve attributes when creating a file.
nolock	Disables NFS locking for compatibility with outdated servers.
port=*n*	Sets the NFS server port number. 2049 is the default.
Option	Function
posix	Runs in a POSIX–compatible mode.
retrans=*n*	Sets the number of retransmissions before a major timeout occurs. The default is 3.
retry=*n*	Sets the amount of time to retry mount. Defaults to 10,000 minutes.
rsize=*n*	Sets the size of the read buffer. The default is 1024 bytes.
soft	Allows the access to time out if the server doesn't respond.
tcp	Runs over TCP instead of UDP.
timeo=*n*	Sets the length of time that occurs before an access times out. Must be used with soft.
udp	Runs over UDP. This is the default.
wsize=*n*	Sets the write buffer size. The default value is currently 1024 bytes.

The fifth field is used by the dump command to determine which filesystems should be backed up. If this field contains a 0, dump will not back up the filesystem. If it contains a 1, dump will back up the filesystem every time dump is run. The dump command, of course, has its own syntax and commands. However, marking certain filesystems in the fstab file with a 0 prevents dump from unnecessarily searching filesystems, such as /proc, that should never be backed up.

The sixth and last field in each entry is used by fsck to determine the order in which filesystems should be checked. If the sixth field contains a 0, the filesystem is not checked. The filesystem is checked first if it contains a 1, second if it contains a 2, and so on. This permits you to define the order in which the systems are checked so that critical systems are checked before less–critical ones when fsck runs during startup. The root partition is usually checked first.

fsck also allows you to specify that filesystems should be checked in parallel to speed the process. For example, assume that you have two different partitions of similar size on two different disks. If you placed a 2 in the sixth field for both of these partitions, they would both be checked second. However, if you specify parallel checking, do it only for partitions that are on different physical disks, and only for those that are of similar size. Otherwise, you get no real performance gain.

NFS entries in the fstab file are exactly like the others, except for the type field, which contains the keyword nfs and the large number of possible NFS mount options. Any of the Linux mount

command–line options listed in Table 9.1, as well as all of those listed in Table 9.2, can be used in an NFS entry in the fstab file.

Most of these options are rarely used. The key options are those that can improve NFS performance, such as tuning the buffer size. The default of 1024 works well for rsize and wsize, but increasing the buffer size can improve performance.

If you have an unreliable server or network, you may want to change the hard failure setting to soft, or at least add the intr option. That way if the server fails, it will not cause your client system to hang on a hard failure; or if it does, you can escape with a keyboard interrupt.

Note All of the options mentioned here can be used on the mount command line with the –o argument. Additionally, all of the options mentioned earlier in the mount command section can also be used in the fstab file. We divided the list of options to show where certain options are most commonly used and to make the list more manageable.

Understanding what to put in the fstab file for an NFS entry can be confusing with all of these options. An easy way to learn what should be in an fstab entry is to mount the filesystem and then display its entry in the mtab file. The mtab file stores information about currently mounted filesystems. The entries in mtab are very similar to those in the fstab file. Simply mount a file and display mtab. Then you'll know what you should enter in fstab. Listing 9.7 is an example.

Listing 9.7: A Sample */etc/mtab* File

```
# mount owl:/home/jane /home/owl
# cat /etc/mtab
/dev/hda3 / ext3 rw 0 0
none /proc proc rw 0 0
/dev/hdd1 /home ext3 rw 0 0
/dev/hdd2 /var ext3 rw 0 0
/dev/hda1 /win vfat rw 0 0
none /dev/shm tmpfs rw 0 0
none /dev/pts devpts rw,gid=5,mode=620 0 0
/dev/cdrom /mnt/cdrom iso9660 ro,nosuid,nodev 0 0
owl:/home/jane /home/owl nfs,addr=172.16.8.15 rw 0 0
```

Based on this, you would add the following entry to the fstab file to mount the /home/owl directory at each boot:

```
owl:/home/jane /home/owl nfs,addr=172.16.8.15 rw 0 0
```

The fstab file is used to automatically remount filesystems at boot time. The automounter automatically mounts NFS filesystems only when they are actually needed.

Automounter

There are two automounter implementations available for Linux: one based on the Berkeley automounter daemon (amd), and one based on the Solaris automounter (automount). Both are configured in a similar manner: Both are given mount points and map files that define the characteristics of the file systems mounted on those mount points. Although its mount points can be defined in / etc/amd.conf, amd mount points and map files are often defined on the command line:

```
amd -a /amd /mnt/wren /etc/nfs/wren.map
```

This command line tells amd that its working directory, in which the real physical mounts are made, is /amd. The mount point for the user directory is /mnt/wren. Like any NFS mount point, this directory must already exist. The map file that defines the filesystem that is mounted on / mnt/wren is named /etc/nfs/wren.map.

This section covers automount in more detail than amd. It does so for two reasons. First, our sample Red Hat 7.2 Linux system runs the autofs script by default, but it will only run the amd script if you enable it. These chkconfig commands from our sample Red Hat system show the following:

```
$ chkconfig --list amd
amd        0:off  1:off  2:off  3:off  4:off  5:off  6:off
$ chkconfig --list autofs
autofs     0:off  1:off  2:off  3:on   4:on   5:on   6:off
```

Second, I personally use automount because the syntax and structure more closely resemble the original Sun implementation, which I have used for years. This makes it easier to move configuration files between systems and to use Sun skills to implement a Linux server. To learn more about amd, see the book *Linux NFS and Automounter Administration* (Sybex, 2001). Written by Erez Zadok, who has been the amd maintainer for the last five years, it is the most comprehensive source of information about amd.

For automount, the mount points are usually not defined on the automount command line. Instead, automount mount points are defined in the auto.master file, which lists all of the maps that are used to define the automatically mounted filesystems. The format of entries in the auto.master file is

```
mount-point     map-name        options
```

The *options* are standard mount command options. Generally, they are not defined here. Most administrators put options in the map file. Most auto.master entries contain only a mount point and a map file. To reproduce the amd configuration shown previously, the auto.master file would contain

```
/mnt/wren      /etc/nfs/wren.map
```

This entry tells automount to mount the filesystem defined in the map file /etc/nfs/wren.map at / mnt/wren whenever a user refers to this directory. On Red Hat systems, the automount daemon is started by the /etc/rc.d/init.d/autofs script. Run autofs reload to load automounter after you change the file.

Note In addition to using the autofs script to reload the automounter configuration, you can run autofs start to start automount, and run autofs stop to shut it down.

The map file contains the information needed to mount the correct NFS filesystem. The format of a map file entry is

```
key     options   filesystem
```

For NFS entries, the *key* is a subdirectory name. When anyone accesses this subdirectory, the filesystem is mounted. The *options* are the standard mount command options, and *filesystem* is the pathname of the filesystem being mounted. The map file for our example contains

```
doc      -ro      wren:/usr/local/doc
man      -ro      wren:/usr/local/man
```

This tells the automounter to mount wren:/usr/doc with read−only permission when any user tries to access /mnt/wren/doc, and to mount wren:/usr/local/man read−only if anyone accesses /mnt/wren/man.

Automounting Home Directories

So far, you've seen automounter handle some simple NFS examples, but it can do much more. For instance, it can automatically mount the home directory of any user based on a single entry in the map file. Assume you have this entry in the auto.master file:

```
/home/owl       /etc/nfs/home.map
```

With this entry, any request for a subdirectory in /home/owl will cause the automounter to check /etc/nfs/home.map for a directory to mount.

Assume that the entry in home.map is

```
*               owl:/home/&
```

This entry can mount any home directory found in the /home directory exported by owl to a like−named mount point on the local host. The * is a wildcard character that tells automounter to match anything the user types in. The & is a special character, which is replaced by the value typed in by the user. For instance, if the user tries to access /home/owl/daniel, automounter mounts owl:/home/daniel. If the user enters /home/owl/kristin, it mounts owl:/home/kristin. When constructing a path to the remote filesystem, automounter uses whatever value the user enters as the "key." This is a very useful feature, particularly when users log in to multiple systems and need access to their home directories from those systems.

NFS is a useful server for your Unix and Linux clients. But on most networks, the majority of clients are Microsoft Windows PCs. Those clients can be made to run NFS with optional client software. However, a better and more natural file−sharing service for Microsoft Windows PCs is provided by Samba. Samba is an implementation of the NetBIOS and Server Message Block (SMB) protocols for Linux.

Understanding SMB and NetBIOS

Microsoft Windows printer− and file−sharing applications are based on NetBIOS (Network Basic Input Output System). The BIOS defines the applications interface used to request DOS I/O services. NetBIOS extends this with calls that support I/O over a network. Developed 20 years ago for the PC Network product sold by Sytek, the NetBIOS API outlived the original product to become part of Windows for Workgroups, LAN Manager, Windows 95/98/ME, and Windows NT/2000.

Originally, NetBIOS was a monolithic protocol that took data all the way from the application to the physical network. NetBIOS has changed over time into a layered protocol. Its layers include the NetBIOS API, the SMB protocol, and the NetBIOS Frame (NBF) protocol.

Today, NetBIOS runs over TCP/IP, which allows NetBIOS applications to run over large internets. It does this by encapsulating the NetBIOS messages inside TCP/IP datagrams. The protocol that does this is NetBIOS over TCP/IP (NBT), which is defined by RFCs 1001 and 1002.

NBT requires some method for mapping NetBIOS computer names, which are the addresses of a

NetBIOS network, to the IP addresses of a TCP/IP network. There are three methods:

IP broadcast A packet containing a NetBIOS computer name is broadcast, and when a host sees its own name in such a broadcast, it returns its IP address to the source of the broadcast.

lmhosts file A file that maps NetBIOS computer names to IP addresses.

NetBIOS Name Server (NBNS) A NBNS maps NetBIOS names to IP addresses for its clients. The Samba nmbd daemon can provide this service.

The systems on an NBT network are classified according to the way they resolve NetBIOS names to IP addresses. There are four possible classifications:

b−node A system that resolves addresses through broadcasts is a *broadcast−node* (b−node). Broadcasting is effective only on a physical network that supports broadcasts, and is usually limited to a single subnet.

p−node A system that directly queries an NBNS name server to resolve addresses is a *point−to−point−node* (p−node).

m−node A system that first uses broadcast address resolution and then falls back to an NBNS server is a *mixed−node* (m−node). Using a "dual approach" eliminates the complete dependence on an NBNS server that is the weakness of the p−node solution. The problem with m−node is that it uses the least−desirable broadcast approach first. In practice, m−nodes are very rarely used.

h−node A system that first attempts to resolve the address using the NBNS server; then falls back to using broadcasts; and if all else fails, looks for a local lmhosts file is a *hybrid−node* (h−node). h−node is the method used by most systems.

NetBIOS Name Service

Even though installing the Samba software has not yet been discussed, this is a good place to discuss the NetBIOS Name Server daemon (nmbd) and how it is configured. nmbd is the part of the basic Samba software distribution that turns a Linux server into an NBNS server. nmbd can handle queries from Windows 95/98/ME/NT/2000 and LanManager clients, and it can be configured to act as a WINS server.

Note The Microsoft implementation of NetBIOS name service is Windows Internet Name Service (WINS). Samba is compatible with WINS and can be used as a WINS server.

nmbd WINS configuration options are defined in the smb.conf file, which is covered in detail later. The key options that relate to running WINS are as follows:

wins support Set to yes or no. This option determines whether or not nmbd runs as a WINS server. no is the default, so by default, nmbd provides browsing controls but does not provide WINS service.

dns proxy Set to yes or no. This option tells nmbd to use DNS to resolve WINS queries that it cannot resolve any other way. This is only significant if nmbd is

running as a WINS server. The default is yes. DNS can help with NetBIOS name resolution only if NetBIOS names and DNS hostnames are the same.

wins server Set to the IP address of an external WINS server. This option is useful only if you're not running a WINS server on your Linux system. This option tells Samba the address of the external WINS server to which it should send NetBIOS name queries.

wins proxy Set to yes or no. The default is no. When set to yes, nmbd resolves broadcast NetBIOS name queries by turning them into unicast queries and sending them directly to the WINS server. If wins support = yes is set, these queries are handled by nmbd itself. If wins server is set instead, these queries are sent to the external server. The wins proxy option is needed only if clients don't know the address of the server or don't understand the WINS protocol.

Provide your clients with the NetBIOS name server's address through DHCP. See the "NetBIOS Options" section in Chapter 8, "Desktop Configuration Servers," for the DHCP options that define a client's NetBIOS configuration. To define the address of the NBNS server, enter the following line in the dhcpd.conf file:

```
option netbios-name-servers 172.16.5.1 ;
```

The NetBIOS name server is generally started at boot time with the following command:

```
nmbd -D
```

When started with the –D option, nmbd runs continuously, listening for NetBIOS name service requests on port 137. The server answers requests using registration data collected from its clients and the NetBIOS name–to–address mappings it has learned from other servers. If the –H /etc/lmhosts option is added to the command line, the server also answers with the mappings defined in the lmhosts file. (You can call this file anything you wish, but the traditional name is lmhosts.)

The lmhosts file is there so that you can manually provide address mapping for the server when it is necessary, though it usually isn't. Most WINS servers do not need an lmhosts file because the servers learn address mappings dynamically from clients and other servers. NetBIOS names are self–registered; clients register their NetBIOS names with the server when they boot. The addresses and names are stored in the WINS database; wins.dat. lmhosts is only a small part of the total database.

The *lmhosts* Files

The lmhosts file contains static–name–to–address mappings. The file is similar to the hosts file described in Chapter 4, "Linux Name Services." Each entry begins with an IP address that is followed by a hostname. However, this time the hostname is the NetBIOS name. Listing 9.8 is a sample lmhosts file.

Listing 9.8: A Sample *lmhosts* File

```
$ cat /etc/lmhosts
172.16.5.5       crow
172.16.5.1       wren
172.16.5.2       robin
172.16.5.4       hawk
```

Given this lmhosts file, the NetBIOS name robin maps to the IP address 172.16.5.2. Notice that these NetBIOS names are the same as the TCP/IP hostnames assigned to these clients. You should always use the same hostnames for your systems for both NetBIOS and TCP/IP. Doing otherwise limits your configuration choices and creates confusion.

NetBIOS name service is an essential part of a NetBIOS network, but the real point of creating such a network is to share files and other network resources. The remainder of this chapter discusses installing and configuring Samba to do just that.

Installing Samba

Samba services are implemented as two daemons. The SMB daemon (smbd), the heart of Samba, provides the file– and printer–sharing services. The NetBIOS Name Server daemon (nmbd) provides NetBIOS–to–IP–address name service.

You can download Samba software from the Internet if you need to. Go to http://www.samba.org/ to select the nearest download site, and then download the file samba–latest.tar.gz from that site. Unzip and untar the source tree into a working directory. Change to that directory, run ./configure, and then run make and make install. This will install the latest version of Samba in /usr/local/samba. However, compiling your own copy of Samba should not be necessary on a Linux system.

Samba is included in most Linux distributions, and can be installed during the initial system installation. Selecting Samba during the Red Hat installation installs the Samba package and the /etc/rc.d/init.d/smb script, which can be run at boot time to start both smbd and nmbd.

If Samba wasn't installed during the initial installation, use rpm or gnorpm to install the software now. If you use gnorpm, the path to the Samba package on a Red Hat system is System Environment/Daemons. Figure 9.2 shows a gnorpm query of the Samba package.

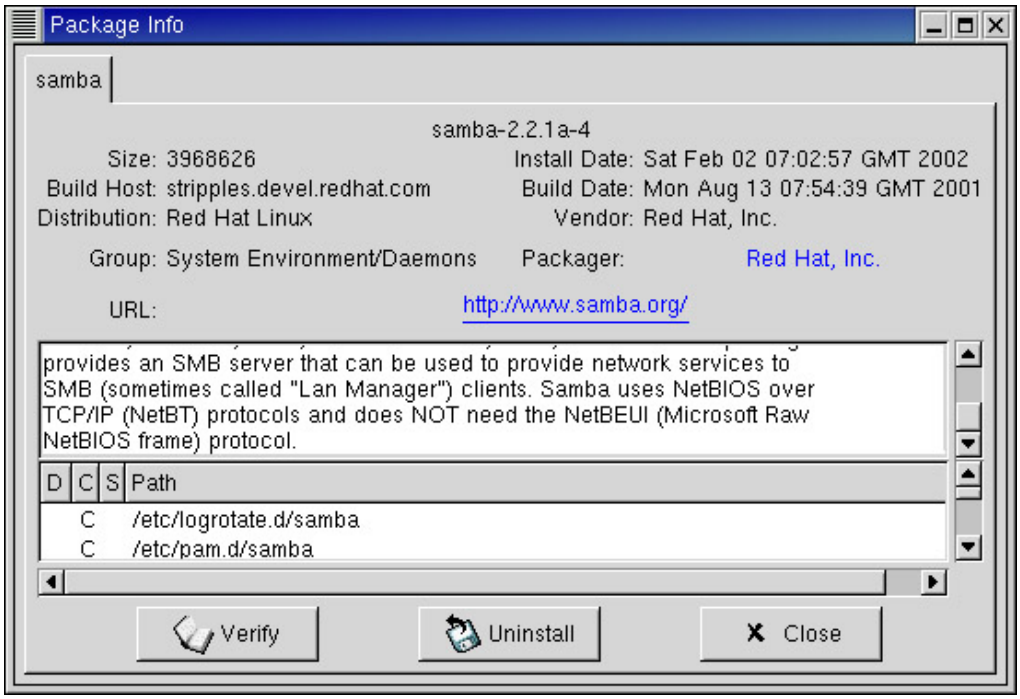


Figure 9.2: The Red Hat Samba RPM

After the software is installed, use chkconfig or tksysv to ensure that the proper scripts run at startup time. On our sample Red Hat system, the script is smb, and we used these chkconfig commands to make sure the script will run at startup:

```
[root]# chkconfig --list smb
smb   0:off   1:off   2:off   3:off   4:off   5:off   6:off
[root]# chkconfig --level 35 smb on
[root]# chkconfig --list smb
smb   0:off   1:off   2:off   3:on    4:off   5:on    6:off
```

The first chkconfig command shows that, even though the Samba RPM is installed on this system, smb was not configured to run at startup. The second chkconfig command causes smb to execute for both runlevel 3 and runlevel 5—the default multiuser run levels.

Next, you run the /etc/init.d/smb script to start the daemons, as follows:

```
[root]# service smb start
Starting SMB services: [  OK  ]
Starting NMB services: [  OK  ]
[root]# service smb status
smbd (pid 5341) is running...
nmbd (pid 5346) is running...
```

As the process IDs in the status response show, the smbd and nmbd daemons are running. Because Red Hat systems won't run the daemons unless an smb.conf file exists, that is a pretty good hint that the Red Hat system comes with Samba pre–configured.

Configuring a Samba Server

The Samba server is configured by the smb.conf file. Look in the startup script to see where smbd expects to find the configuration file. On a Red Hat system, it is /etc/samba/smb.conf, on another system it might be /etc/smb.conf, and the default used in most Samba documentation is /usr/local/samba/lib/smb.conf. Use find or check the startup script so you know where it is on your system.

The smb.conf file is divided into sections. Except for the global section, which defines configuration parameters for the entire server, the sections are named after shares. A *share* is a resource offered by the server to the clients. In the context of this chapter, it is a filesystem that is offered by the server for the clients to use for file sharing. A share can also be a shared printer. Printer sharing is the topic for Chapter 10, "Printer Services," so Samba printer shares are covered there.

The best way to learn about the smb.conf file is to look at one. Minus comments and all of the lines that deal with sharing printers, the Red Hat smb.conf file contains the active lines shown in Listing 9.9.

Listing 9.9: Active Lines in the Red Hat *smb.conf* File

```
[global]
   workgroup = MYGROUP
   server string = Samba Server
   log file = /var/log/samba/%m.log
   max log size = 0
   security = user
   encrypt passwords = yes
   smb passwd file = /etc/samba/smbpasswd
```

```
   socket options = TCP_NODELAY SO_RCVBUF=8192 SO_SNDBUF=8192
   dns proxy = no
[homes]
   comment = Home Directories
   browseable = no
   writable = yes
   valid users = %S
   create mode = 0664
   directory mode = 0775
```

The *smb.conf* Variables

Reading an smb.conf file can be confusing if you don't understand the variables found in the file. Table 9.3 lists each variable and the value it carries.

Table 9.3: smb.conf Variables

Variable	Meaning
%a	Client machine architecture
%d	Server process ID
%g	GID of the username assigned to the client
%G	GID of the username requested by the client
%h	DNS hostname of the server
%H	Home directory of the username assigned to the client
%I	IP address of the client
%L	NetBIOS name of the server
%m	NetBIOS name of the client
%M	DNS hostname of the client
%N	NIS server if NIS is supported
%p	NIS home directory if NIS is supported
%P	The root directory of the current service
%R	The protocol negotiated during connection
%S	The name of the current service
%T	The date and time
%u	The username assigned to the client
%U	The username requested by the client
%v	The Samba version number

Variables provide flexibility because each variable is replaced in the configuration by a value obtained from the system. This allows the same configuration statement to be interpreted differently in different situations. Here is an example from the Red Hat smb.conf file in Listing 9.9:

```
log file = /var/log/samba/%m.log
```

In this example, the %m variable is replaced by the NetBIOS name of the client, so a different log file is created for each client using the client's NetBIOS name. If the NetBIOS name of the client is crow, the log file is named /var/log/samba/crow.log. If the client's NetBIOS name is robin, the log file is /var/log/samba/robin.log.

The *smb.conf* Global Section

The Red Hat sample configuration file contains two sections: *global* and *homes*. The global section defines several parameters that affect the entire server. Minus the parameters that are specific to printer sharing, which are covered in Chapter 10, the parameters in the Red Hat smb.conf global section are the following:

workgroup Defines the workgroup of which this server is a member. A workgroup is a hierarchical grouping of hosts. It organizes network resources in the same way that directories organize file resources, and it is used for the same reason: Grouping computers into workgroups helps a user locate related systems. Workgroups are not used for security. Hosts that are not in the workgroup are still allowed to share files with systems that are. Replace the MYGROUP name in the example with a meaningful workgroup name of 15 characters or less.

server string Defines the descriptive comment for this server. The string is displayed by the net view command on Windows clients, so it provides an opportunity for you to describe the server. Change the string in the example to something meaningful for your system.

log file Defines the location of the log file. The most interesting thing about this entry is that it contains an smb.conf variable.

max log size Defines the maximum size of a log file in kilobytes. The default is 5MB, or 5000KB. If the maximum size is exceeded, smbd closes the log and renames it with the extension .old. The Red Hat configuration sets this to 0, which means "unlimited"—there is no maximum log size.

security Defines the type of security used. In Samba, there are four possible settings:

share Requests share–level security. This is the lowest level of security. Essentially, a resource configured with share security is shared with everyone. It is possible to associate a password with a share, but the password is the same for everyone who wants to use that share.

user Requests user–level security. Every user is required to enter a username and an associated password. By default, this is the username and password defined in /etc/ passwd that the user would use to log in to the Linux server. The default values for passwords can be changed, as we'll see in a minute.

server Defines server–level security. This is similar to user–level security, but an external server is used to authenticate the username and password. The external server must be defined by the password server option.

domain Defines domain–level security. In this scheme, the Linux server joins a Windows NT domain and uses the Windows NT domain controller as the server that approves usernames and passwords. Use the password server option to point to the Windows NT Primary

Domain Controller (PDC). Log in to the PDC, and create an account for the Linux system. Finally, add these lines to the global section on the Linux system:

```
domain master = no
local master = no
preferred master = no
ostype = 0
```

encrypt passwords Specifies whether or not Samba should use encrypted passwords. Setting this parameter to yes makes the server more compatible with the Windows clients, and it makes it harder for intruders to sniff passwords from the network. Encrypted passwords are both more secure and more compatible. If this parameter is set to no, clear–text passwords are used, which require changes in the client configurations. See the sidebar "Clear–Text Passwords" for information on the Registry changes that are needed to make Windows clients compatible with clear–text passwords.

smb passwd file Use this parameter to point to the location of the smbpasswd file. When encrypted passwords are used, the Samba server must maintain two password files: passwd and smbpasswd. Use the mksmbpasswd.sh script to build the initial smbpasswd file from the passwd file.

socket options Defines performance–tuning parameters. The sample Red Hat configuration sets TCP_NODELAY to tell Samba to send multiple packets with each transfer, which is actually the default. The SO_RCVBUF and SO_SNDBUF options set the send and receive buffers to eight kilobytes, which may slightly increase performance. A detailed study of Samba performance tuning is beyond the scope of this book. See Chapter 12 of *Linux Samba Server Administration* by Rod Smith (Sybex, 2000) for a good discussion of Samba performance tuning.

dns proxy Specifies whether or not nmbd should forward unresolved NBNS queries to DNS, as described in the previous section. The Red Hat configuration sets this to no, so it does not send unresolved NBNS queries to DNS.

Clear–Text Password

When Samba uses clear–text passwords, no password database synchronization is required because only one database, /etc/passwd, is used. However, clear–text passwords are not compatible with many versions of Windows because those versions require encrypted passwords. To force those clients to use clear–text passwords, you must edit the Registry of every client. For Windows 95/98/ME, the Registry setting is

```
[HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD\VNETSUP]

"EnablePlainTextPasswords"=dword:00000001
```

On Windows NT, the setting is:

```
[HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\
ÄRdr\Parameters]

"EnablePlainTextPasswords"=dword:00000001
```

On Windows 2000, the setting is

```
[HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\
ÄLanmanWorkStation\Parameters]

"EnablePlainTextPasswords"=dword:00000001
```

Clear−text passwords are not as secure as encrypted password. Additionally, maintain two databases on a Linux server is usually simpler than editing the Registry on every Windows client. For this reason, most administrators find encrypted passwords are better and less of a headache.

Beyond changing the workgroup name, the Red Hat system runs with the sample configur−ation without changes to any of these global parameters. The other section in the sample configuration file that relates to file sharing is the homes section.

The *smb.conf* Homes Section

The homes section is a special share section. It tells smbd to permit users to access their own home directories through SMB. Unlike the other share sections covered later, the homes section does not tell smbd the specific path of the directory being shared. Instead, smbd uses the home directory from the /etc/passwd file based on the username of the user requesting the share. It is this special section that makes a user's home directory on the server available to the user on their PC. The homes section from the Red Hat example is

```
[homes]
   comment = Home Directories
   browseable = no
   writable = yes
   valid users = %S
   create mode = 0664
   directory mode = 0775
```

The configuration parameters defined in this homes section are the following:

comment Provides a description of the share that is displayed in the comment field of the Network Neighborhood window when this share is viewed on a Microsoft Windows system.

browsable Specifies whether or not all users may browse the contents of this share. no means that only users with specific permission (that is, the correct user ID) are allowed to browse this share. yes means all users, regardless of UID, can browse the share. This parameter only controls browsing; actual access to the contents of the share is controlled by standard Linux file permissions.

writable Specifies whether or not files can be written to this share. If yes, the share can be written to. If no, the share is read−only. This parameter defines the actions permitted by Samba. Actual permission to write to the directory defined by the share is still controlled by standard Linux file permissions.

valid users Defines the users who are allowed to use this share. In the Red Hat example, the service name (%S) is used.

create mode Defines the file permission used when a file is created in this share. (See the discussion of file permission earlier in this chapter.)

directory mode Defines the directory permissions used when a directory is created in this share.

Both the global and homes sections described are included in the sample Red Hat configuration. Having an understanding of the elements used to create those sections, you're ready to create your own share section in the smb.conf file.

Sharing a Directory through Samba

To share a directory through Samba, create a share section in smb.conf that describes the directory and the conditions under which you are willing to share it. To share the /home/sales directory used in the NFS examples and a new directory named /usr/local/pcdocs, you might add the two share sections shown in Listing 9.10 to the sample smb.conf file.

Listing 9.10: Samba File Shares

```
[pcdocs]
     comment = PC Documentation
     path = /usr/local/pcdocs
     browsable = yes
     writable = no
     public = yes

[sales]
     comment = Sales Department Shared Directory
     path = /home/sales
     browsable = no
     writable = yes
     create mode = 0750
     hosts allow = .sales.foobirds.org
```

Each share section is labeled with a meaningful name. This name is displayed as a folder in the Network Neighborhood window on client PCs. Each section contains some commands you have already seen and a few new commands. The first new command is path, which defines the path of the directory being offered by this share.

The pcdocs share also contains the command public. public allows anyone to access the share, even if they don't have a valid username or password. These public users are granted "guest account" access to the share. On a Linux system, this usually means they run as user nobody and group nobody, and are limited to world permissions.

Setting File and Directory Permissions

The sales share is being offered as a writable share. The create mode command controls the permissions used when a client writes a file to the /home/sales directory. In the sample in Listing 9.10, it is specified that files will be created with read/write/execute for the owner, read/execute for the group, and no permissions for the world (750). A related command, directory mode, defines the permission used when a client creates a directory within a share. For example:

```
directory mode = 0744
```

This sets the permissions for new directories to read/write/execute for the owner, read/execute for the group, and read/execute for the world (744). This is a reasonable setting that allows cd and ls to work as expected because directories must have the world execute bit set in order for the change directory (cd) command to work properly.

Limiting Access to a Share

The sales share section also contains a hosts allow command, which defines the clients that are allowed to access this share. Even if a user has the correct username and password, they are allowed to access this share only from the specified hosts. By default, all hosts are granted access, and specific access is controlled by the username and password.

The hosts identified in the hosts allow command in Listing 9.10 are identical to those listed in the NFS example. This illustrates that Samba can also control access with domain wildcards.

There are several different ways to define individual hosts or groups of hosts in the hosts allow command. As the name of the command implies, it uses the same syntax as the hosts.allow file discussed in Chapter 12, "Security." Some examples of how it can be used in the smb.conf file are as follows:

> **hosts allow = 172.16.5.0/255.255.255.0** Allows every host on network 172.16.5.0 access to the share.
>
> **hosts allow = 172.16. EXCEPT 172.16.99.0/255.255.255.0** Allows every host on network 172.16.0.0 to have access to the share, except for those hosts on subnet 172.16.99.0. 172.16 might be the enterprise network, and 172.16.99 might be an untrusted subnet where publicly accessible computers are located.

In addition to the hosts allow command, there is a hosts deny command that defines computers that are explicitly denied access to the share. Its syntax is similar to that of the hosts allow command.

Combining these two new share sections with the section that came with the Red Hat configuration creates a server that does everything you want. It provides access to user home directories. It provides access to public directories used to offer online documentation or other publicly shared resources. And it offers private directories that are accessible only to members of the selected group. This provides everything that NFS did in a manner that is much simpler for Microsoft Windows clients to use.

Of course, you're not limited to serving only Windows clients. Linux systems can also be Samba clients.

Using a Linux Samba Client

NFS is the most common way to share files between Linux systems, and features such as autofs and mounting from fstab make integrating NFS into a Linux client very seamless. But not all servers are Linux servers. It is possible that you will need to configure a Linux system as a client to a Windows NT/2000 server, or even as a peer to a Windows desktop. For those situations, use the Samba client tools.

Using *smbclient*

The smbclient program is a tool for transferring files with a system offering an SMB share. It is particularly useful for transferring files with Windows systems that do not have FTP server software. smbclient acts like an FTP tool for SMB share files. Listing 9.11 illustrates this.

Listing 9.11: Using *smbclient*

```
$ smbclient //robin/temp -W sybex
added interface ip=172.16.5.2 bcast=172.16.5.255 nmask=255.255.255.0
Password:
smb: \> ls al*_0.jpg
  alana1_0.jpg           A     6147  Sun Jul  8 11:39:42 2001
  alana2_0.jpg           A     8180  Sun Jul  8 11:46:56 2001
  alana3_0.jpg           A    23296  Wed Aug  8 09:37:24 2001
  alana4_0.jpg           A    42857  Sun Nov 25 16:50:42 2001
  alana5_0.jpg           A    22456  Sun Nov 25 16:53:00 2001
  alana6_0.jpg           A    55847  Wed Feb  6 16:00:20 2002
  alana7_0.jpg           A    42799  Wed Feb  6 16:10:50 2002

  51795 blocks of size 131072. 11861 blocks available
smb: \> get alana1_0.jpg
getting file alana1_0.jpg as alana1_0.jpg (average 158.0 kb/s)
smb: \> quit
```

The smbclient tool is invoked by the smbclient command. The share that you're accessing is described on the command line using the Microsoft Uniform Naming Convention (UNC). The UNC format is //*server*/*sharename,* where *server* is the NetBIOS name of the server, and *sharename* is the name of the share.

If a share password is required, which might be the case if the server uses share–level security, it follows the UNC on the command line. In the Listing 9.11 example, share–level security is used, but the password is not provided on the command line, so the server will prompt the user for it.

Use the –U command option, and provide the username and password separated by a % if the server uses user–level security. If a workgroup name is required, provide it with the –W option, as shown in Listing 9.11.

After the username and password have been provided, files are sent and retrieved using exactly the same commands as FTP. If you can use FTP, you can use smbclient.

The smbclient program is the workhorse of the Samba client tools. It is not very elegant, but it is the basis for several other client tools that are shell scripts that use smbclient to get the work done. A more graceful way to integrate SMB server files into the Linux filesystem is with smbfs.

Using *smbmount*

The SMB filesystem (smbfs) allows you to mount SMB shares, and use them as if they were part of the Linux filesystem. For this to work, the kernel must support the smbfs filesystem. Listing 9.12 shows a quick check for kernel support of smbfs.

Listing 9.12: Checking /*proc*/*filesystems*

```
[root]# modprobe smbfs
```

```
[root]# cat /proc/filesystems
nodev   proc
nodev   sockfs
nodev   tmpfs
nodev   shm
nodev   pipefs
        ext2
        iso9660
nodev   devpts
        ext3
        vfat
nodev   autofs
nodev   smbfs
```

The /proc pseudo filesystem provides a glimpse into the kernel. The pseudo file /proc/filesystems lists the filesystems that are in the kernel, either because they are compiled in or configured as loadable modules. Of course, loadable modules are not loaded unless they are needed, so Listing 9.11 starts with a modprobe command to force smbfs to load. If smbfs does not load, the kernel may need to be reconfigured as described in Chapter 13.

Shares can be mounted using the smbmount command and dismounted using the smbumount command. Listing 9.13 mounts the same share shown in the smbclient example by using smbmount.

Listing 9.13: An *smbmount* Example

```
[root]# ls /home/robin
[root]# smbmount //robin/temp /home/robin workgroup=sybex
Password:
[root]# ls /home/robin/alana*_0.jpg
/home/robin/alana1_0.jpg  /home/robin/alana4_0.jpg  /home/robin/alana7_0.jpg
/home/robin/alana2_0.jpg  /home/robin/alana5_0.jpg
/home/robin/alana3_0.jpg  /home/robin/alana6_0.jpg
```

The ls command at the start of Listing 9.13 shows that /home/robin is an empty directory, which we will use as a mount point (the location within the Linux filesystem at which the share is mounted). The smbmount command starts with the name of the share written in Microsoft UNC format. The share name is followed by the name of the Linux mount point. A list of options follows the mount point. In this case, workgroup is the only option required.

After it is mounted, the share has essentially the same look and feel of any Linux directory, and most standard Linux commands can be used to manipulate the files in the share directory. (Listing 9.13 shows an ls of certain files in the directory.) Of course, not everything is the same on Linux systems and Windows systems. Some features of the Linux filesystem are not available from all SMB servers. For example, a Windows 95/98/ME system offering a share does not have file–level security, nor does it understand Linux UIDs and GIDs. smbfs does its best to "fake it." It uses the UID and GID in force when smbmount was started. You can override these defaults with command–line options. For example, the following tells smbfs to use the UID 689, presumably the UID assigned to tyler; and the GID 100, which is the GID of users:

```
smbmount //crow/user/tyler /home/tyler/crow \
    username=tyler password=Wats?Watt? uid=689 gid=100
```

To dismount an SMB share, use the smbumount command with the path of the mount point:

```
[root]# smbumount /home/robin
```

Shares can also be mounted from the fstab file using the mount command with the –a and the –t smbfs arguments. Red Hat uses this technique to mount Samba shares in the netfs startup script. To do this, the Samba share must be defined in the fstab file. For example, to mount the Samba share used in Listing 9.13, the following entry could be added to the fstab file:

```
//robin/temp    /home/robin    smbfs   workgroup=sybex  0 0
```

Here, the filesystem name is the UNC–formatted name of the remote share, and the filesystem type is smbfs. When the mount command is invoked with –t smbfs, it transfers control to the program /sbin/mount.smbfs. On a Red Hat system, /sbin/mount.smbfs is just a symbolic link to smbmount. The values in the first four fields of the fstab entry are passed to smbmount for processing.

smbfs makes using SMB shares on a Linux client much easier than accessing those shares through smbclient.

In Sum

File sharing is the foundation application of departmental networks. Linux servers make excellent platforms for file servers—Linux is fast and very stable, and it provides a wider choice of file services than most other server operating systems.

Files are commonly shared in three different ways:

- Through direct login. Users wanting to share files can directly log in to a Linux server, regardless of the capability of their desktop systems. Files can then be shared using the Linux filesystem.
- Through Network File System. NFS is the leading file–sharing protocol of Unix systems. Linux systems come with a full range of NFS server and client software.
- Through Server Message Block protocols. SMB is the file–sharing protocol used by Microsoft Windows systems. A Linux system can act as an SMB server or client to share files with Microsoft Windows systems.

In the next chapter, we configure Linux as a departmental print server. Again, you will see that Linux has the capability to integrate both Unix and Windows clients on a single network.

Chapter 10: Printer Services

Printer servers allow everyone on a network to share printers. Linux printer servers offer a few different techniques for sharing printers. This chapter covers two techniques that support two different groups of clients. The traditional Unix network technique uses the Line Printer daemon (lpr) command and an lpd server. This approach is best suited for serving Linux and Unix clients. The other technique uses a Samba server to share printers with Microsoft Windows clients. But before you can use any technique to share a printer with your clients, you must install and configure the printer on your server.

Installing Printers

Printer installation is part of the initial system installation on some Linux distributions. Others, such as Red Hat Linux 7.2, wait until the system is running before configuring the printer. Configuring the printer during or after the initial installation is essentially the same procedure. To install a printer, you must know the type of printer and its capabilities. All examples in this section use printconf, which is a printer–configuration tool available as part of Red Hat Linux 7.2. But the basic concepts of printer configuration are the same for all Linux distributions.

To launch the printconf tool on a Red Hat system, enter printconf–tui for a text version of the tool or printconf–gui for an X Window System version. In these examples, we use the X printconf tool.

The first time you run printconf, no printer queues are listed because you haven't configured any yet. Click the New button to start a wizard that helps you define a printer queue. After an introductory screen, the wizard displays the window shown in Figure 10.1.

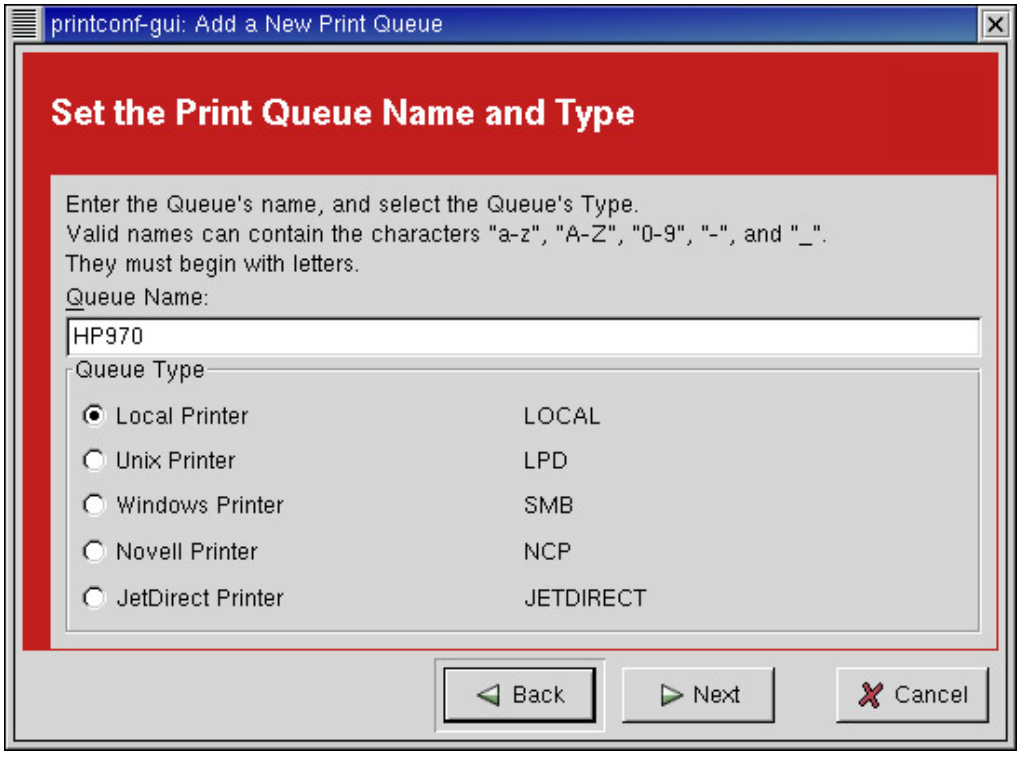


Figure 10.1: Selecting a print queue type

In the Queue Name box at the top of the window, enter the name by which this printer will be known on the local system (for example, HP970). This window also gives you five choices of how the system will communicate with the printer. These are called the *queue types*. The choices are the

following:

Local Printer A local printer is directly attached to one of the physical ports of the Linux system. If your system is a printer server, you have at least one printer directly attached to the computer.

Unix Printer A remote printer attached to an lpd server. This selection configures the client side of lpd printer sharing. The server for the remote printer doesn't really need to be a Unix system. For example, many network–connected printers support lpd printing natively, allowing you to offer printer service without configuring a separate computer to act as a print server. Many different systems provide lpd servers. In this chapter, you configure your Linux system to be an lpd server.

Windows Printer The SMB protocol is used to communicate with this printer. This setting selects the client side of SMB printer sharing, which allows your Linux system to use the printers shared by Microsoft Windows systems. Of course, SMB servers don't have to be Microsoft Windows systems, as you saw in Chapter 9, "File Sharing." Later in this chapter, we configure a Linux system to be an SMB printer server.

Novell Printer The Novell NetWare Core Protocol (NCP) is used to communicate with this printer. This setting selects the client side of NetWare printer sharing, which allows the Linux system to use printers offered by Novell servers.

JetDirect Printer A JetDirect printer is one that is directly attached to the network and uses the HP JetDirect printer protocol.

In Figure 10.1, Local Printer is selected, which is a common selection for a printer server because printers are directly attached to a print server. Printers are often directly attached through parallel ports, and they are in our sample system. Our sample Red Hat system defines four parallel printer ports, as the ls in Listing 10.1 demonstrates.

Listing 10.1: Listing the Printer Ports

```
$ ls -l /dev/lp*
crw-rw----    1 root     lp     6,  0 Aug 30  2001 /dev/lp0
crw-rw----    1 root     lp     6,  1 Aug 30  2001 /dev/lp1
crw-rw----    1 root     lp     6,  2 Aug 30  2001 /dev/lp2
crw-rw----    1 root     lp     6,  3 Aug 30  2001 /dev/lp3
```

The Linux device names for the parallel ports are /dev/lp0, /dev/lp1, /dev/lp2, and /dev/lp3. The numbers listed just before the dates are the device's major and minor numbers. The major number (6) stands for the parallel port driver. The minor number (0, 1, 2, or 3) indicates which addressable device it is for that driver.

Selecting Local Printer in the window shown in Figure 10.1 causes the printconf tool to scan the parallel printer ports. printconf displays a window that lists the ports it believes are active. On our sample system, printconf displays only one active port, /dev/lp0, as Figure 10.2 shows.

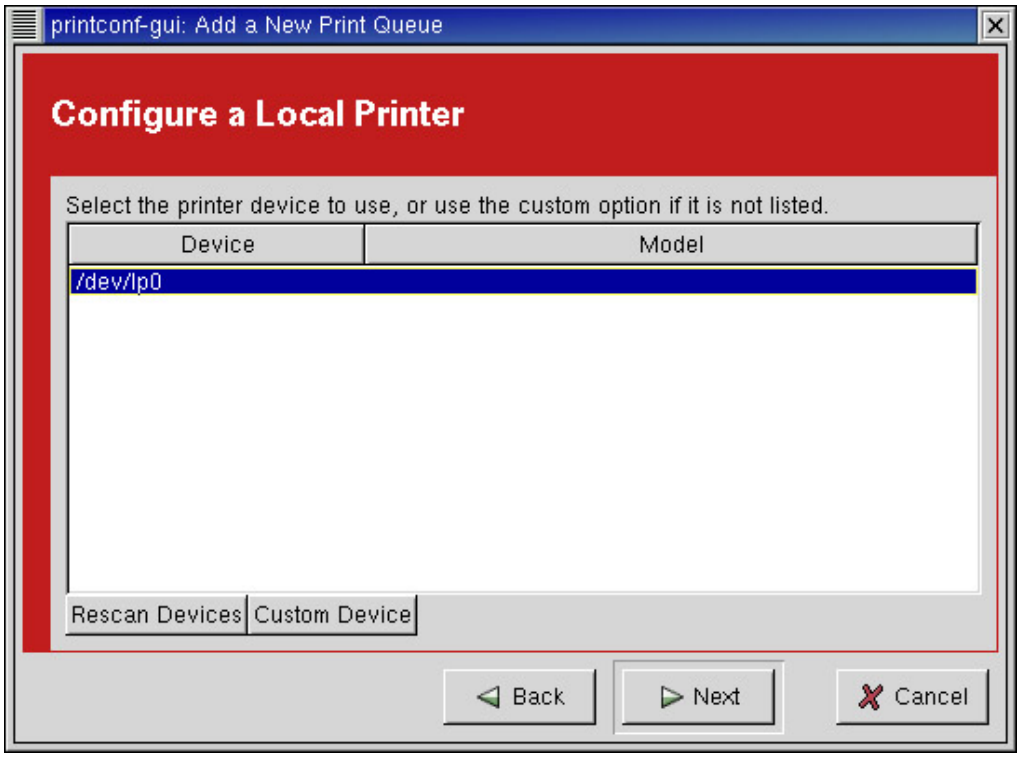


Figure 10.2: The active local printer port

Only one parallel interface was found because only one is installed in this small system. If you think the system missed an interface that really is there, click the Rescan Devices button to force printconf to scan the interfaces again. Alternatively, you can click the Custom Device button, and type in the name of the device to which the printer is attached.

Next, you're asked to select the printer driver. A printer driver does more than just drive the device. It is also the program that prepares the print file for the specific printer. Every type of printer has its own command language and data format language. For example, one printer might use Adobe's PostScript, and another might use HP's Printer Command Language (PCL). Traditionally, Unix systems used only PostScript or text printers; and if you're buying a new printer, selecting a PostScript printer is still a good choice. A PostScript printer will work with every Unix and Linux system in a seamless manner. However, many Linux systems do not have a PostScript printer attached. PostScript printers are more expensive than similar printers that do not offer PostScript. Many Linux systems are built with lower–cost printers. To ensure that a wide variety of printers will work, Linux provides many printer drivers. The window shown in Figure 10.3 shows a printer driver being selected.

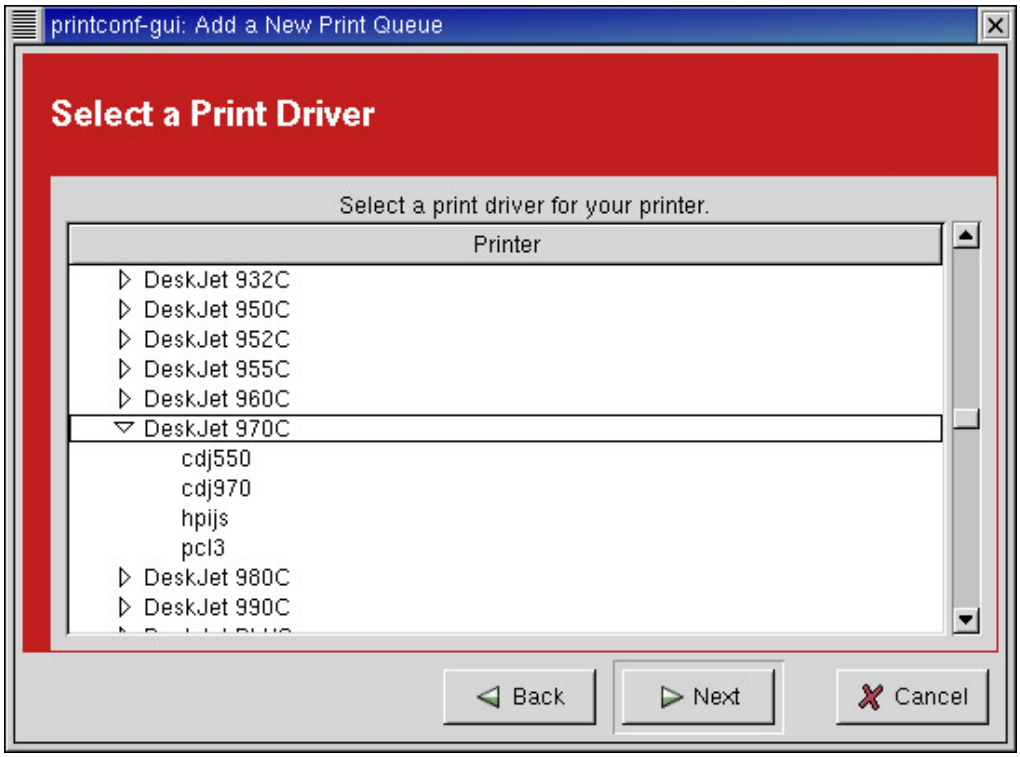


Figure 10.3: Selecting a printer driver

Use the scroll bar to view the list of printer drivers. Beyond the PostScript, text–only, and raw selections at the top of the list, the drivers are organized by printer manufacturer and printer model. Select your printer to display the list of drivers available for that printer. Some printers have only a single driver available, but many printers have a choice of possible drivers, depending on the features you want to use. Figure 10.3 shows that the HP DeskJet 970c has four possible driver choices: cdj550 (Color DeskJet 550 driver), cdj970 (Color DeskJet 970 driver), hpijs (HP InkJet Series driver), and pcl3 (Printer Command Language 3 driver).

In this case, we pick the cdj970 driver because it was designed for this specific printer. But the choice is not always clear. Go to http://www.linuxprinting.org/ for information about each driver. If you decide to change the driver or any other setting, you can do it by editing an existing printer configuration.

Tip It's a very good idea to use one of the printers for which the system has a driver. Trying to make a printer work that is not compatible with your system is a lot of work for very little reward. Use a printer from the driver list.

Selecting a printer driver is the last step of the installation. After the printer is installed, it is added to the list of printers in the main printconf window; you can then highlight the printer and click the Default button to make it your default printer, or highlight the printer and click Edit to change the configuration. When the Edit button is selected, the window in Figure 10.4 appears.

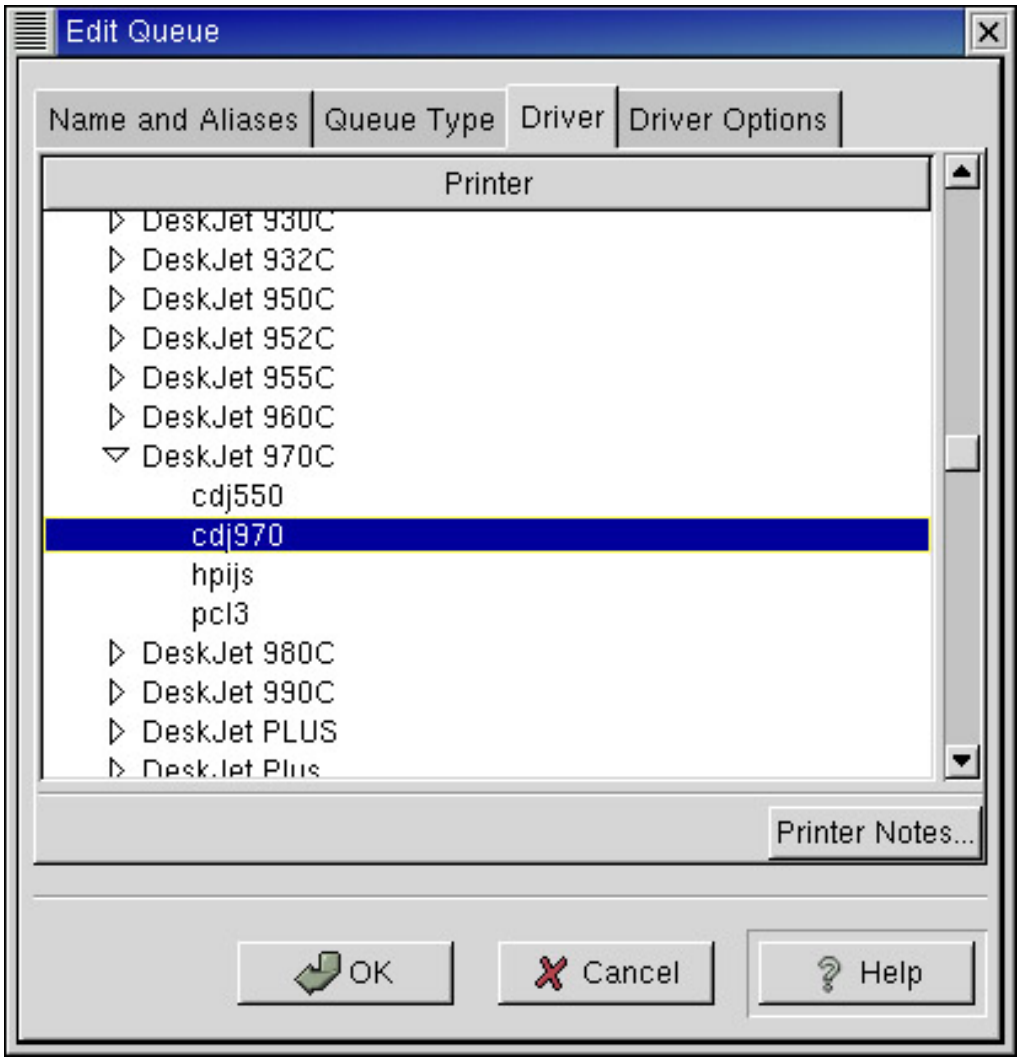


Figure 10.4: Editing a printer configuration

The Edit Queue window contains four tabs:

Name and Aliases This tab displays the Queue Name that was entered when the printer was installed; and it allows you to add, change, and delete aliases for the queue name. A printer can be known by more than one name. This is the tab you use to assign multiple names to a printer.

Queue Type The Queue Type tab holds a drop–down box from which any of the five queue types can be selected. The currently selected queue type is displayed along with the configuration parameters associated with that queue type. For example, in this section, we installed a local printer. For the local printer queue type, the device name is displayed along with three buttons: Rescan Devices, Custom Device, and Autoselect Driver. Rescan Devices and Custom Device were explained earlier. Clicking on Autoselect Driver causes the system to probe the printer attached to the device in an attempt to determine the correct driver for the printer. For remote printer queue types, this tab is more complex. We configure remote printers in the next section.

Driver This tab shows the same driver selections we saw in Figure 10.3, with the current driver selection highlighted. It also presents the Printer Notes button, which displays information about the printer and the driver from http://www.linuxprinting.org/ when clicked.

Driver Options The last tab displays the optional configuration parameters available for the selected driver. Selecting a different driver in the Driver tab changes the layout and parameters displayed on the Driver Options tab. The choices for the parameter settings are not always obvious, particularly if you don't have the documentation for the printer. Go to the printer manufacturer's web page and get the spec sheet for the printer, and go to http://www.linuxprinting.org/ for detailed information about the driver before you change the parameter settings.

After installing and configuring the printer, test it by selecting a test from the Test menu at the top of the printconf window. The tests include both the ASCII and the PostScript test pages, and various types of images. Select tests that suit the capabilities of the printer and that test the uses to which the printer will be put.

Configuring Remote Printers

Remote printers require slightly more configuration than local printers. Figure 10.5 shows a configuration window for a remote SMB printer; Figure 10.6 shows one for a remote Unix printer.

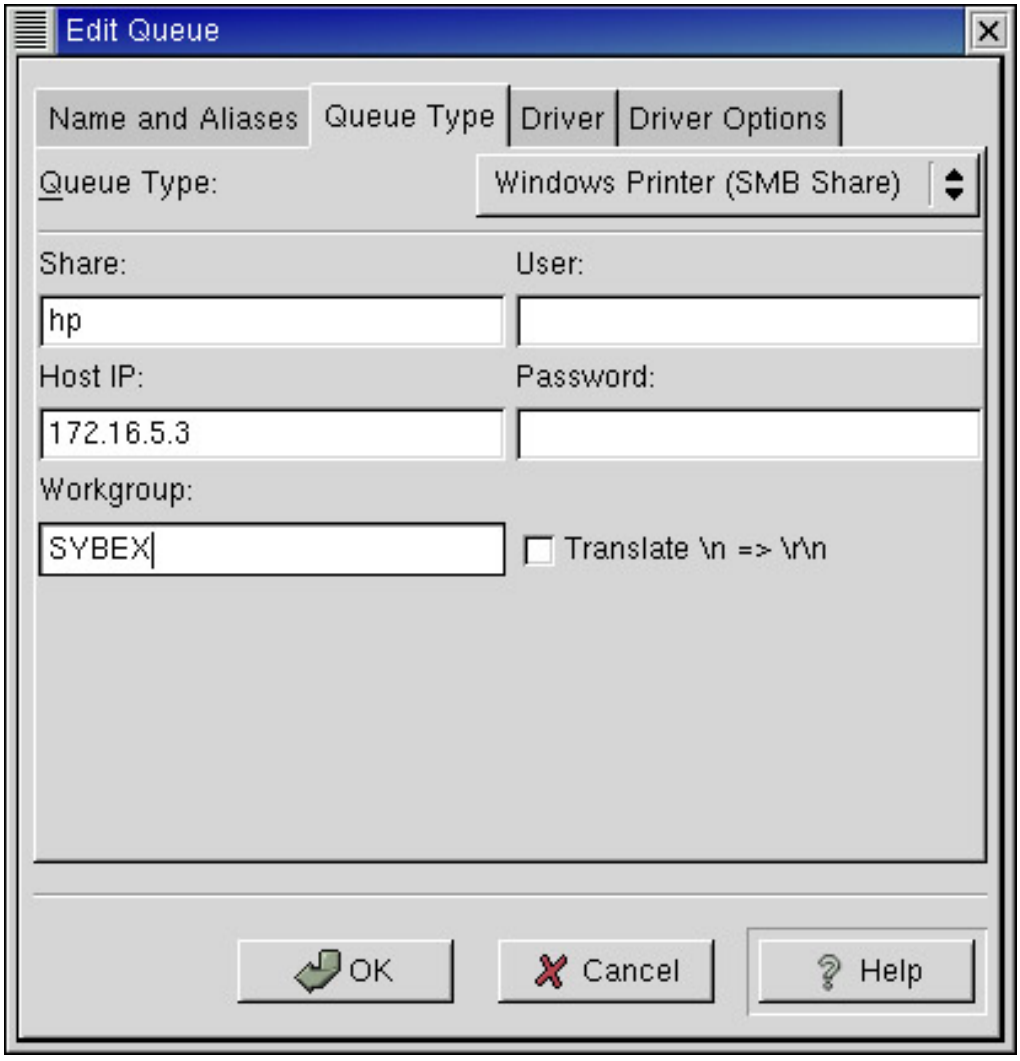


Figure 10.5: Configuring a remote SMB printer

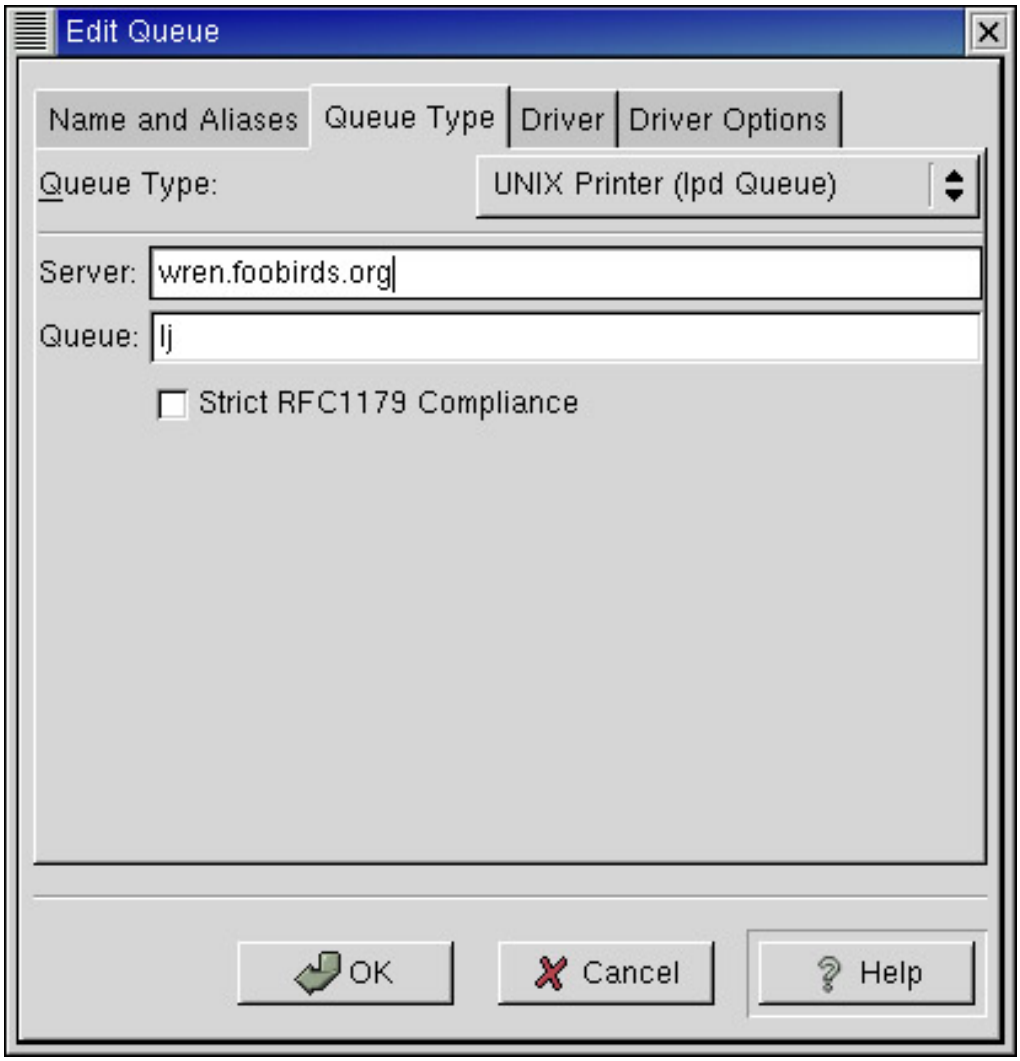


Figure 10.6: Configuring a remote Unix printer

The Queue Type tab for a remote Unix printer has two fields to define the remote computer and the printer on that computer. The Server box requires the hostname of the printer server; the name must be written in a form that your system can resolve to an IP address. The Queue box requires the name of the remote printer as it is defined on the remote server. Note that this doesn't need to be the same as the name that you use locally as the queue name field. The Queue Name value defined on the Name and Aliases tab is your local name for the printer; the Queue value defined on the Queue Type tab is the remote server's name for the printer.

The SMB printer entry in Figure 10.5 has five fields to define the remote printer:

Share Defines the SMB share name of the printer. This is the name that the remote server advertises for the printer, which is the same name you see when you browse the remote server.

Host IP Specifies the IP address of the server.

Workgroup Specifies the name of the workgroup to which the print server belongs. In the example, the workgroup is named SYBEX.

User Specifies your SMB username on the remote server. This is needed only if the server requires user–level security to access its printers.

Password Specifies the password required by the remote server to access its printers. This is combined with the username for user–level security, but it may be required even when the User field is empty by a server that has share–level security applied to the printer.

In addition to the five dialog boxes, the SMB printer configuration window has a check box for translating each line–feed character to a carriage return plus a line–feed. This is sometimes necessary when dealing with systems that run Windows software.

Figures 10.5 and 10.6 show configuration examples for remote Unix and SMB printers. Yet there are two more types of remote printers that can be used: JetDirect printers and Novell printers. Selecting JetDirect Printer from the queue type causes the system to ask for the IP address of the remote printer and the port number used to communicate with the printer. By default, JetDirect printers listen to port 9100, which should not normally be changed. TCP/IP networking and the basic Linux print commands are all you need to communicate with a JetDirect printer. Novell printers are slightly more complex.

To use a remote Novell printer, you must have the proper NetWare software installed on your system. On our sample Red Hat system, it means that you must install the RPM for the NetWare Core Protocol File System (ncpfs), and you should have the CONFIG_NCP_FS option set when you compile your Linux kernel. If printconf finds nprint (the Novell printer tool from the ncpfs package), you're asked to provide the name of the Novell server, the name of the printer on that server, your Novell username, and your Novell password on the Queue Type tab.

Clearly, in order to successfully configure a printer, you must know the make and model of the printer, what its capabilities are, and how it is connected to the system. Gather this information before you start the configuration process. Although none of this is difficult, it is not obvious unless you have the right information on hand.

Understanding *printcap*

The /etc/printcap file defines the printers and their characteristics. Understanding a printcap file is the most difficult part of configuring a Linux printer server. There are an enormous number of possible configuration parameters, and the syntax of many of the parameters is terse and arcane. The complexity of the printcap file is one reason why tools such as printconf were created.

Generally, you use a tool to configure a printer. You don't directly edit the printcap file. In fact, if you ever want to create your own printcap entries, you should place them in /etc/printcap.local, which is a file that is included into printcap during processing. Despite the fact that it is unlikely you will manually build a printcap file, a system administrator should understand the content of the file well enough to read it and get a general idea of what it is doing. To do that, you need a basic understanding of the file structure and the configuration parameter syntax.

The file contains one entry for each active printer. Printer entries can span multiple lines by ending a line with a backslash (\) to indicate that the following line is a continuation line, and beginning the continuation line with a vertical bar (|) or a colon (:). Every field in a printer entry, other than the printer name, begins and ends with a colon.

Each printer entry starts with a printer name. Multiple printer names can be used if they are separated by vertical bar characters. Traditionally, one printer was always named lp, and many configurations continue to require that one printer has the name lp.

Every active line in the printcap file begins with a printer name, a vertical bar, or a colon. Comments begin with a hash mark (#). Blank lines and leading blank characters are ignored.

printcap Parameters

The configuration parameters used in a printcap file define the printer characteristics that lpd needs to know in order to communicate with the printer. The syntax of the parameters varies slightly, depending on the type of value they are assigned. There are three types of parameters:

Boolean All printcap boolean values default to false. Specifying a boolean enables its function. Booleans are specified simply by entering the parameter name in the file. For example, :ab: tells lpd to always print banners.

Numeric Some parameters are assigned numeric values. The syntax of numeric parameters separates the value from the parameter name with a #. For example, :mx#1000: sets the maximum size for an acceptable print file to 1MB.

String Some parameters use string values. The syntax of string parameters separ–ates the value from the parameter name with an =. For example, :rp=laser: defines the name of a remote printer as laser.

A glance at the man page shows that there are a large number of printcap parameters. Thankfully, you'll never need to use most of them. Most printer definitions are fairly simple, and most printcap files are small. Servers usually have only one or two directly attached printers; any other printers defined in the printcap are probably remote printers.

A Sample *printcap*

Below is the printcap file that is the result of defining three printers using the printconf tool. One is a remote SMB printer, one is a remote Unix printer, and the other is a locally attached printer. Listing 10.2 shows the printcap file that printconf created when these three printers were configured.

Listing 10.2: A Sample *printcap* File

```
# /etc/printcap
#
# DO NOT EDIT! MANUAL CHANGES WILL BE LOST!
# This file is autogenerated by printconf-backend during lpd init.
#
# Hand edited changes can be put in /etc/printcap.local, and will be included.

lp|Canon:\
        :sh:\
        :ml=0:\
        :mx=0:\
        :sd=/var/spool/lpd/lp:\
        :af=/var/spool/lpd/lp/lp.acct:\
        :lp=/dev/lp0:\
        :lpd_bounce=true:\
        :if=/usr/share/printconf/util/mf_wrapper:

laser:\
        :sh:\
        :ml=0:\
        :mx=0:\
        :sd=/var/spool/lpd/laser:\
```

```
        :af=/var/spool/lpd/laser/laser.acct:\
        :rm=172.16.5.15:\
        :rp=lj:\
        :lpd_bounce=true:\
        :if=/usr/share/printconf/util/mf_wrapper:

hp:\
        :sh:\
        :ml=0:\
        :mx=0:\
        :sd=/var/spool/lpd/hp:\
        :af=/var/spool/lpd/hp/hp.acct:\
        :lp=|/usr/share/printconf/util/smbprint:\
        :lpd_bounce=true:\
        :if=/usr/share/printconf/util/mf_wrapper:

#######################################################
## Everything below here is included verbatim from  ##
## /etc/printcap.local                              ##
#######################################################
# printcap.local
#
# This file is included by printconf's generated printcap,
# and can be used to specify custom hand edited printers.
```

The first printer, which has the printer name lp, is the one that is directly attached to the parallel port. This printer is also known by the name Canon. The printer's names are followed by the configuration parameters that define exactly how the printer is used. The parameters are

sh Specifies whether or not headers or banner pages are printed. This is the "suppress headers" boolean. By default, it is false, meaning that headers are not suppressed. In other words, headers are printed by default. printconf, however, sets the boolean to true. (Simply specifying a boolean sets it to true.) This means that headers and banner pages are not printed between print jobs. This setting is fine for a small system, but you might want to comment out this boolean if you have several users so that an identifying banner page is printed for each print job.

ml Defines the minimum value used to determine a printable character. Setting this to zero turns off this check and attempts to print everything.

mx Defines the maximum acceptable size for a print file. Setting this to 0 means that there is no limit on the size of print jobs.

sd Defines the path to the spool directory in which print jobs are spooled as they wait for the printer.

af Defines the path to the accounting file in which print job statistics are written.

lp Defines the device name for the printer. For a local printer, this is the port to which the printer is attached.

lpd_bounce Tells lpd to process the print file through a filter before passing it to the printer. A filter is the software that prepares the file for a specific printer. This is an LPRng innovation and not supported by the traditional lpr.

if Defines the input filter for this printer. In this case, the input filter is a script file named mf_wrapper (for magic filter wrapper). It takes the information passed to it by lpd and properly formats a call to the program magicfilter–t, which in turn prepares the file for the printer.

The second printer in this file (laser) is a remote printer. The remote machine to which the printer is attached is defined by the rm parameter, and the name of the remote printer on that machine is defined by the rp parameter. In the example, the remote host is identified by the IP address 172.16.5.15, and the remote printer name is lj. Most of the real configuration work for this printer takes place on a remote system. Even though this is a remote printer, it has a local spool directory. Print files are written to the local spool directory where they are queued for delivery to the remote system.

The third printer, known as hp, is the remote SMB printer. This is the most unique definition. lpd naturally handles local printers and remote lpd printers. Remote SMB printers require a little more effort. The real work for this type of printer is handled by the smbprint program, which is invoked through a pipe character (|) with the lp parameter in Listing 10.2. In this case, /usr/share/printconf/util/smbprint is a script that is used to properly format the call to the real smbprint program. (smbprint is covered later in this chapter.) The lp parameter is used to call the smbprint program after the input filter has processed the file. The filtered file is then sent by smbprint to the remote server for printing.

Writing a printcap from scratch is unnecessary. To create a printcap entry, select a printer that is documented to work with your Linux system and use a printer configuration tool. If you need to customize a printer entry, move the entry to printcap.local and make your changes there. Use the printcap manual page to help you read and update the printcap.local. After the printer is configured and running locally, it is ready to share with others.

Sharing Printers with *lpd*

The Line Printer daemon (lpd) provides printer services for local and remote users. It is an essential service that is started at boot time from a startup script. On both Red Hat systems, lpd is started by the /etc/rc.d/init.d/lpd script that is generally included in the startup by default, and can be controlled chkconfig or tksysv.

Use the lpd script to stop, start, or reload the Line Printer daemon. Because the printcap file is read only by lpd during its startup, the reload option is useful to incorporate changes if you edit the printcap file. Here is an example of using the reload command with the start–up script:

```
# service lpd reload
Reloading lpd: [  OK  ]
```

The printcap file is not the only place in which configuration parameters can be set for lpd. Red Hat and many other Linux distributions use the LPRng implementation of lpd, which can read configuration settings from /etc/lpd.conf. Although not all printcap settings can be overridden by lpd.conf, several can. Generally, however, there is no need to customize the lpd.conf file because the default values used by LPRng are correct for most configurations. An example is our sample Red Hat system. It has an lpd.conf file, but that file does not contain a single active entry.

Using *lpr*

Use the Line Printer Remote (lpr) program to send print jobs to the Line Printer daemon. There are several lpr command–line arguments, but the command usually identifies the printer and the file to be printed, as in this example:

```
% lpr -Plaser sample.txt
```

This command sends a file called sample.txt to a printer called laser. The printer can be local or remote; it doesn't matter as long as the printer is defined in the printcap file, and thus is known to lpd. Assuming the printcap shown earlier, laser is a remote printer. If no printer is defined on the command line, lpr attempts to get the value from the environment variables PRINTER, LPDEST, NPRINTER, or NGPRINTER. If none of those variables is defined, lpr uses the first printer defined in the printcap file as the default printer for our sample Red Hat system. Systems that use different implementations of lpr, default to the printer name lp.

Managing *lpd*

The Line Printer Control (lpc) program is a tool for controlling the printers and administering the print queue on lpd printer servers. Table 10.1 lists the lpc commands used on our sample Red Hat 7.2 system and their purposes.

Table 10.1: lpc Commands

Command	Usage
active *printer*[@*host*]	Checks to see if the remote print queue is active.
abort *printer*	Kills a printer daemon.
class *arguments*	Controls classes of print jobs.
client *printer*	Displays the client configuration for the printer.
defaultq	Lists the default queue used by lpc.
defaults	Lists the default configuration settings.
debug *printer level*	Turns on the specified level of debugging for a printer.
disable *printer*	Turns off spooling to a print queue.
down *printer message*	Ends spooling and printing, and outputs a message.
enable *printer*	Enables spooling to a print queue.
exit	Exits from lpc.
help *command*	Displays a description of the command.
hold *printer job*	Holds a job from printing on the printer.
holdall *printer*	Holds all jobs for the printer.
kill *printer*	Aborts and then starts the printer.
lpd *printer*[@*host*]	Retrieves the process ID of lpd from the remote host servicing the printer.
lpq *printer* [*options*]	Runs lpq. (More on lpq later.)
lprm *printer job*	Runs lprm. (More on lprm later.)
move *prt1 job prt2*	Moves a print job from *ptr1* to *ptr2*.
msg *printer text*	Change the status message for *printer* to *text*.
noholdall *printer*	Turns off holdall for the printer.

quit	Exits from lpc.
redirect *prt1 ptr2*	Sends all jobs in print queue *prt1* to print queue *ptr2*.
redo *printer job*	Reprints a job.
release *printer job*	Releases a job to the printer for printing.
reread *printer*[@*host*]	Rereads the printcap configuration for a printer.
server *printer*	Shows the server printcap entry for the printer.
start *printer*	Enables printing and spooling.
status *printer*	Displays the status of printers and queues.
stop *printer*	Stops a printer after the current job ends.
topq *printer job user*	Moves a print job to the top of the queue.
up *printer*	Enables spooling and printing.

printer is the name of the printer or the keyword all, which can be used on commands that apply to all printers. If *printer* is not specified, the default printer is used. *job* is the job number from the print queue. *host* is the name of a remote print server. If it is not provided, lpc uses the server name from the configuration of the specified printer. Use the help command to view the details of the syntax of specific commands before using them for the first time.

lpc can be invoked interactively. Listing 10.3 shows examples of a few simple lpc commands.

Listing 10.3: Using *lpc* Interactively

```
[root]# lpc
lpc>status all
 Printer  Printing Spooling Jobs  Server Subserver Redirect Status/(Debug)
hp@robin   enabled  enabled    0    none    none
lp@robin   enabled  enabled    0    none    none
laser@crow enabled  enabled    0    none    none
lpc>kill lp
Printer: lp@robin
kill server PID 0 with Interrupt
lp@robin.foobirds.org: killed job
lpc>status lp
 Printer  Printing Spooling Jobs  Server Subserver Redirect Status/(Debug)
lp@robin  enabled  enabled    0    none    none
lpc>exit
```

Note that the keyword all is used in place of a printer name in the first sample status command. all can be used to refer to all printers in most lpc commands that accept a printer name as an optional parameter. The status commands used in Listing 10.3 can be used by anyone. But many other lpc commands can be used only by the root user (for example, the kill command, which kills and then restarts a print queue). Despite the name kill, the printer is restarted as the second status command in Listing 10.3 shows.

The lpc command can be used to rush print jobs through a crowd by moving the jobs to the top of the print queue. In Listing 10.4, the root user moves print job 840 to the top of the queue for printer lp.

Listing 10.4: Viewing and Reordering a Print Queue

```
[root]# lpc
lpc>lpq lp
```

```
Printer: lp@robin 'Canon'
 Queue: 3 printable jobs
 Server: pid 1830 active
 Unspooler: pid 1831 active
 Status: waiting for subserver to exit at 11:35:27.386
 Rank  Owner/ID        Class  Job Files       Size Time
 1     root@robin+829     A   829 ndc.txt      344 11:35:00
 2     sara@robin+840     A   840 rev.txt      833 11:35:15
 3     ed@robin+842       A   842 conf.txt     399 11:35:27
lpc>topq lp 840
Printer: lp@robin
lp: selected 'sara@robin+840'
lp@robin.foobirds.org: started
lpc>lpq lp
Printer: lp@robin 'Canon'
 Queue: 3 printable jobs
 Server: pid 1830 active
 Unspooler: pid 1831 active
 Status: waiting for subserver to exit at 11:35:55.158
 Rank  Owner/ID        Class  Job Files       Size Time
 1     sara@robin+840     A   840 rev.txt     833 11:35:15
 2     root@robin+829     A   829 ndc.txt     344 11:35:00
 3     ed@robin+842       A   842 conf.txt    399 11:35:27
lpc>exit
```

In Listing 10.4, the lpq command is used to list the contents of the lp printer queue. The job numbers are listed under the heading "Job" when the queue is displayed. The topq command moves the specified job to the top of the selected print queue. The second lpq command shows that job 840 is now first in line to be printed.

The lpq command shown in Listing 10.4 is really a standalone command; it does not have to be run from within an lpc session. The list of jobs queued for a printer can be requested from the command line. The –P command–line argument selects which printer queue is displayed. Here's an example of displaying the queue for the printer hp:

```
[root]# lpq -Php
Printer: hp@robin
 Queue: 3 printable jobs
 Server: pid 1369 active
 Unspooler: pid 1420 active
 Status: waiting for subserver to exit at 11:28:39.365
 Rank  Owner/ID        Class  Job Files       Size Time
 1     root               A   368 conf.txt    399 11:28:16
 2     alana@robin+395    A   395 ndc.txt     344 11:28:31
 3     tyler@robin+421    A   421 rev.txt     833 11:28:39
```

This is the same queue display that is seen inside of lpc.

lpq is not the only lpc command that can be used outside of lpc. The lprm command is also a standalone program. Use lprm to remove a queued print job. The job can be removed by the owner of the job or by the root user. Assume that alana wants to remove print job number 395 shown in the previous example. She enters the command shown in Listing 10.5.

Listing 10.5: Removing Jobs from the Print Queue

```
$ lprm -Plp 395
Printer lp@robin:
  checking perms 'alana@robin+395'
```

```
dequeued 'alana@robin+395'
```

Sharing Printers with Samba

Chapter 9 shows how a Linux server can be an SMB server by using the Samba software package, and it shows how Samba is used to share files. What Chapter 9 doesn't show is that Samba can also be used to share printers with SMB clients. Here, you see how that's done.

First, of course, you need to make sure that Samba is installed in your system. (See Chapter 9 for those details.) After Samba is installed, shared printers are configured through the smb.conf file.

Defining Printers in the *smb.conf* File

The best way to understand the SMB configuration file is to look at one that works. Red Hat systems come with a preconfigured smb.conf file that includes support for sharing printers. The active lines in the Red Hat smb.conf file are shown in Listing 10.6.

Listing 10.6: *smb.conf* with Printer Sharing

```
[global]
   workgroup = MYGROUP
   server string = Samba Server
   printcap name = /etc/printcap
   load printers = yes
   printing = lprng
   log file = /var/log/samba/%m.log
   max log size = 0
   security = user
   encrypt passwords = yes
   smb passwd file = /etc/samba/smbpasswd
   socket options = TCP_NODELAY SO_RCVBUF=8192 SO_SNDBUF=8192
   dns proxy = no
[homes]
   comment = Home Directories
   browseable = no
   writable = yes
   valid users = %S
   create mode = 0664
   directory mode = 0775
[printers]
   comment = All Printers
   path = /var/spool/samba
   browseable = no
   guest ok = no
   writable = no
   printable = yes
```

You saw many of these lines in Chapter 9, so you already know to change the workgroup option to the correct workgroup name for your network and to change the server string to something meaningful that describes your server. Some of these lines, however, were not covered in that chapter. All of the new lines deal with sharing printers. Three of the new lines are in the global section:

printcap name Defines the location of the printcap file. As you'll see in a minute, the

printcap file is used to identify the printers that are available to share. The default path is / etc/printcap, which is the value set in Listing 10.6.

load printers Tells smbd whether or not it should offer all of the printers in the printcap file as shared printers. The default is yes, which tells Samba to share all of the printers defined in the printcap file. no means do not read the printcap file at all. If no is specified, all shared printers must be defined individually. In Listing 10.6, this is set to yes, so the Red Hat configuration will share every printer defined in /etc/printcap.

printing Identifies the printing system used on the server. Samba supports eight different printing systems identified by these keywords: BSD, AIX, LPRNG, PLP, SYSV, HPUX, QNX, SOFTQ, and CUPS. Select the printing system used by your Linux system. Red Hat Linux 7.2 uses LPRng, which is identified by the keyword lprng in Listing 10.6.

These lines are used to prepare the server for printer sharing and to prepare it to automatically share the printers defined in the printcap file. In addition to these global lines, there is an entire share section labeled "printers." It also deals with automatically sharing printers.

Printers Share Section

The printers section performs a similar function to the homes section, which makes every home directory available to the appropriate user. The printers section is defined to make every printer available to your clients. The Red Hat printers share section is

```
[printers]
   comment = All Printers
   path = /var/spool/samba
   browsable = no
   guest ok = no
   writable = no
   printable = yes
```

You know the comment, browsable, writable, and path options from Chapter 9. Here, however, path does not define the path of a shared file. Instead, it defines the path of the spool directory for the SMB shared printers. But which printers are shared? Based on the two options defined in the global section, all printers that are defined in the printcap file.

There are two lines in this section that you have not seen before. The first is printable, which identifies this share as a printer. The default for this option is no, meaning that by default, shares are considered to be file shares instead of printer shares. When you create a printer share, you must set this option to yes. Enabling printable permits clients to write printer files to the spool directory defined by the path option. This appears to contradict the writable command that says clients cannot write to the share. The writable option is there to say that no one can write a file to the spool that is not a print file. Because print files are created by clients in the spool directory, you may want to add a create mode command that limits the permissions of the files created. For example, create mode = 0700.

The other new line, guest ok, defines whether or not guest accounts are permitted access to the resource. This is exactly the same as the public option discussed in Chapter 9, so these two options are used interchangeably. no means that the user nobody cannot send a print job to the printer. A user must have a valid user account to use the printer. This is designed to prevent guest users from

abusing the printer, but it is also useful to have a valid username for sorting out print jobs if you use banner pages and accounting on your server.

Generally, this section is all you need to make every printer on the server available to all of your clients. You can use the host allow command described in Chapter 9 to restrict access to printers in the same way that you restrict access to files, but in general, a printer server offers all of its printers to all of its clients.

smb.conf Printer Configuration Options

If you don't want to share every printer defined in the printcap file, you can remove the printers section, set the load printers option to no, and add individual share sections for just those printers that you do want to share. Individual share sections can be created for each printer in the same way that they are created for file sharing. In addition to the printer configuration options described previously, you can use any relevant options described in Chapter 9.

An smb.conf file with a share section for a specific printer might contain the following:

```
[global]
   workgroup = SYBEX
   server string = Author's Printer server
   load printers = no
   printing = lprng
   log file = /var/log/samba/%m.log
   max log size = 0
   security = user
   encrypt passwords = yes
   smb passwd file = /etc/samba/smbpasswd
   socket options = SO_RCVBUF=8192 SO_SNDBUF=8192
   dns proxy = no
[homes]
   comment = Home Directories
   browseable = no
   writable = yes
   valid users = %S
   create mode = 0664
   directory mode = 0775
[hp5m]
   comment = PostScript Laser Printer
   path = /var/spool/samba
   browsable = no
   public = no
   writable = no
   create mode = 0700
   printable = yes
   printer = lp
```

In this case, no printers section is included. Instead, a share section named hp5m is added that shares printer lp. The printer name (lp) must be found in the printcap file for this to work. The printcap name parameter is allowed to default to /etc/printcap.

Using an SMB Printer

A Linux system can be an SMB client as easily as it can be an SMB server. A Linux user can print to a remote SMB printer with a standard lpr command if the SMB printer is properly defined in the printcap file. We used the Red Hat printconf tool earlier in the chapter to define a remote SMB printer. The printconf screen used to create that printer was shown in Figure 10.5.

The printconf input in Figure 10.5 created this printcap entry:

```
hp:\
        :sh:\
        :ml=0:\
        :mx=0:\
        :sd=/var/spool/lpd/hp:\
        :af=/var/spool/lpd/hp/hp.acct:\
        :lp=|/usr/share/printconf/util/smbprint:\
        :lpd_bounce=true:\
        :if=/usr/share/printconf/util/mf_wrapper:
```

The key to making this entry work is the pipe to the smbprint script in the lp parameter used for this printer. smbprint is a script file that uses smbclient to print to the remote system.

The additional information entered in the printconf window is used to configure the smbprint script. That information is stored in the script.cfg file in the spool directory of the SMB printer. A glance at the sd parameter in the printcap file tells the path of the spool directory, and a cat command shows the contents of the script.cfg file. Listing 10.7 shows this.

Listing 10.7: The *script.cfg* File for a Samba Printer

```
$ cat /var/spool/lpd/hp/script.cfg
share='hp'
hostip='172.16.5.3'
user=''
password=''
workgroup='SYBEX'
translate='no'
```

The script.cfg file contains six entries, which must occur in exactly the order shown. If no value is provided for an entry, the entry still appears in the file, but with a null entry, e.g., password=''. The six entries are:

share Defines the share name, which is the share name of the remote printer.

hosTip Specifies the numeric IP address of the remote server.

user Defines the user's login name on the remote server.

password Defines the password required by the remote server for access to the printer.

workgroup Identifies the workgroup to which the print server belongs.

translate Specifies whether or not line–feed characters will be translated to carriage–return/line–feed pairs.

If the printcap is properly configured, and the script.cfg file contains the necessary information, a user on our sample Linux system should be able to print to the printer on 172.16.5.3 by entering **lpr –Php sample.txt.** Most users find this to be the easiest way to use a remote SMB printer from a Linux system.

However, you can also use a remote printer directly through the smbclient software. Using smbclient to access shared files was discussed in Chapter 9, and printing with smbclient is very similar. Use smbclient in the same way to connect to the share, except this time the share is a remote printer. For example:

```
smbclient //parrot/hp -W SYBEX
```

After connecting to the printer share, use the same put command that you use to transfer a file to the server. When smbclient transfers a file to a printable share, the file is printed. For example, to print the file sample.txt on the hp printer after connecting to that printer with the smbclient command, enter the following:

```
smb:> put sample.txt
```

For a book–length treatment of Samba and sharing printers through Samba, see *Linux Samba Server Administration* by Rod Smith (Sybex, 2000).

In Sum

File and printer sharing are the basic services of a departmental network. Linux is an excellent platform for providing these network services because it can provide the native services expected by Unix clients in addition to the same services in the native format expected by Microsoft Windows clients. Other departmental servers do not do as effective a job of integrating all of your clients together into a single network.

Another service that Linux excels at is e–mail. The next chapter concludes the section on departmental servers with a discussion of departmental mail services.

Chapter 11: More Mail Services

Overview

A departmental mail server usually acts as a mailbox server that holds mail for its clients until they are ready to download it for their readers. The mailbox service supports mobile users and systems that do not receive mail in real time. Two commonly used techniques that Linux offers for creating a mailbox server are Post Office Protocol (POP) and Internet Message Access Protocol (IMAP). This chapter uses both protocols to configure a Linux system to act as a department mailbox server.

In addition to the essential mailbox services, Linux systems offer some additional mail services that you may want to use. These include spam filters and tools that are designed to help you limit the amount of unwanted junk mail that bombards your users.

Understanding POP and IMAP

sendmail, which was covered in Chapter 5, "Configuring a Mail Server," provides support for SMTP—the standard mail transport protocol for TCP/IP networks. When sendmail is configured to run as a daemon, it listens to the network and collects incoming mail. That mail can be forwarded, as directed by the /etc/aliases file, or stored locally in the user's mailbox in /var/mail. If the mail is stored locally on the server, the user can log in to the server and read the mail there. However, most users prefer to read their mail on their personal computers. Yet they don't want to run sendmail on those systems to collect mail in real time because of the complexity of sendmail and because they want to be able to take those systems offline at any time. DNS, sendmail, and the mailbox protocols combine to give the users what they want. MX records in DNS route mail to the mail server. sendmail, running as a daemon, collects and stores the mail on the server. The mailbox protocols move the mail from the server to the user's system when the user is ready to read the mail. POP and IMAP are the two protocols most commonly used to move mail from the server to the user's computer.

The POP Protocol

There are two versions of POP: POP2 and POP3. The POP protocols verify the user's login name and password, and move the user's mail from the server to the user's local mail reader. Both protocols perform the same basic functions, but they are incompatible. POP2 uses port 109, and POP3 uses port 110. Linux systems come with both versions of POP, but POP2 is rarely used. Most POP clients use POP3.

POP3 is defined in RFC 1939, "Post Office Protocol—Version 3." It is a simple request/ response protocol. The client sends a command to the server, and the server responds to the command. Table 11.1 shows the set of POP3 commands defined in RFC 1939.

Table 11.1: POP3 Commands

Command	Function
USER *username*	The username required for the login.
PASS *password*	The user's password required for the login.
STAT	Requests the number of unread messages/bytes.

RETR *msg*	Retrieves message number *msg*.
DELE *msg*	Deletes message number *msg*.
LAST	Requests the number of the last message accessed.
LIST [*msg*]	Requests the size of message *msg* or of all messages.
RSET	Restores deleted messages, and resets the message number to 1.
TOP *msg n*	Prints the headers and the first *n* lines of message number *msg*.
NOOP	Does nothing except request an OK response from the remote server.
APOP *mailbox string*	Identifies a mailbox, and provides an MD5 digest string for authentication. Used as an alternative to USER/PASS.
UIDL [*msg*]	Requests the unique ID for the specified message number, or a listing of unique IDs for all messages.
QUIT	Ends the POP3 session.

The POP3 protocol is simple enough to be done by hand over a telnet connection. Listing 11.1 shows a sample POP3 session that demonstrates the function of several of the protocol commands.

Listing 11.1: Using the POP Protocol with *telnet*

```
$ telnet localhost 110
Trying 127.0.0.1...
Connected to ani.foobirds.org.
Escape character is '^]'.
+OK POP3 ani.foobirds.org v2000.70rh server ready
USER craig
+OK User name accepted, password please
PASS Wats?Watt?
+OK Mailbox open, 4 messages
STAT
+OK 4 8184
LIST
+OK Mailbox scan listing follows
1 1951
2 1999
3 2100
4 2134
.
RETR 1
+OK 1951 octets
... an e-mail message 1951 bytes long ...
.
DELE 1
+OK Message deleted
QUIT
+OK Sayonara
Connection closed by foreign host.
```

The first three lines after the telnet command (Trying, Connected, and Escape) are output from the telnet command, as is the very last line (Connection closed). All of the other lines in Listing 11.1 are POP3 commands and responses. Positive responses start with the string +OK, which indicates that the command executed successfully. When a command fails, the response begins with the string –ERR. The first +OK response in Listing 11.1 is in reply to the connection request from telnet. The response indicates that the POP server is ready.

The user then logs in with the USER and PASS commands. The username and password provided here must match a valid username and password found in the /etc/passwd file. Notice that the password is sent as clear text. POP3 provides a more secure MD5 login mechanism through the APOP command.

The STAT command and the LIST command are used to inquire about the messages stored in the mailbox. STAT shows that there are four messages with a combined length of 8184 bytes. The LIST command shows the size of each individual message in the mailbox. When it comes to mail, size matters because the system downloading the mail needs to know it has sufficient disk space to store it. In Listing 11.1, all of the mail messages are small, so storage is not an issue.

The first message from the mailbox is downloaded with the RETR 1 command. It is then removed from the server mailbox with the DELE 1 command. Normally, messages are retrieved in order, and are deleted after they are retrieved, but they don't have to be. When using telnet to input the POP commands, you're in complete control. You can download messages out of order, you don't need to delete the messages you download, and you don't need to download messages you delete. Deleting messages instead of downloading them is often very useful. On occasion, a corrupted or overly large message stored on the server causes download problems for the desktop client. From the client, the user can log on via telnet, and delete the offending message to get everything running normally again.

Of course, a POP connection is not normally run manually over a telnet connection. This is done here only to illustrate the function of the protocol. Only telnet to the POP port for testing. Using your knowledge of the protocol and the configuration, you can telnet to the POP port, and test whether your server responds. The telnet test proves that the daemon is available, installed, and ready to run.

The IMAP Protocol

Internet Message Access Protocol (IMAP) is an alternative to POP. It provides the same basic service as POP, and adds features to support mailbox synchronization. *Mailbox synchronization* is the ability to read individual mail messages on a client or directly on the server while keeping the mailbox on both systems completely up–to–date.

On an average POP server, the entire contents of the mailbox are moved to the client, and either deleted from the server or retained as if never read. Deletion of individual messages on the client is not reflected on the server because all of the messages are treated as a single unit that is either deleted or retained after the initial transfer of data to the client. IMAP provides the ability to manipulate individual messages on the client or the server, and to have those changes reflected in the mailboxes of both systems.

Like the POP protocol, IMAP is also a request/response protocol with a small set of commands. Table 11.2 lists the basic set of IMAP commands from version 4 of the IMAP protocol.

Table 11.2: IMAP4 Commands

Command	Use
CAPABILITY	Lists the features supported by the server.
NOOP	Literally means "No Operation," but sometimes used as a way to poll for new messages or message status updates.
LOGOUT	Closes the connection.

AUTHENTICATE	Requests an alternative authentication method.
LOGIN	Opens the connection, and provides the username and password for plain–text authentication.
SELECT	Opens a mailbox.
EXAMINE	Opens a mailbox as read–only.
CREATE	Creates a new mailbox.
DELETE	Removes a mailbox.
RENAME	Changes the name of a mailbox.
SUBSCRIBE	Adds a mailbox to the list of active mailboxes.
UNSUBSCRIBE	Deletes a mailbox name from the list of active mailboxes.
LIST	Displays the requested mailbox names from the complete set of all available mailboxes.
LSUB	Displays the requested mailbox names from the set of active mailboxes.
STATUS	Requests the status of a mailbox.
APPEND	Adds a message to the end of the specified mailbox.
CHECK	Forces a checkpoint of the current mailbox.
CLOSE	Closes the mailbox, and removes all messages marked for deletion.
EXPUNGE	Removes from the current mailbox all messages that are marked for deletion.
SEARCH	Displays all messages in the mailbox that match the specified search criteria.
FETCH	Retrieves a message from the mailbox.
STORE	Modifies a message in the mailbox.
COPY	Copies the specified messages to the end of the selected mailbox.
UID	Searches for or fetches messages based on the message's unique identifier.

This command set is more complex than the one used by POP because IMAP does more. The protocol is designed to remotely maintain mailboxes that are stored on the server, and the protocol commands clearly illustrate the "mailbox" orientation of IMAP. Despite the increased complexity of the protocol, it is still possible to run a simple test of your IMAP server using telnet and a small number of the IMAP commands. Listing 11.2 shows this.

Listing 11.2: Testing IMAP with *telnet*

```
$ telnet localhost imap
Trying 127.0.0.1...
Connected to ani.foobirds.org.
Escape character is '^]'.
* OK [CAPABILITY IMAP4 IMAP4REV1 STARTTLS LOGIN-REFERRALS
ÄAUTH=LOGIN] ani.foobirds.org IMAP4rev1 2000.287rh
Äat Mon, 6 May 2002 17:36:57 -0400 (EDT)
a0001 login craig Wats?Watt?
a0001 OK LOGIN completed
a0002 select inbox
* 3 EXISTS
* 0 RECENT
* OK [UIDVALIDITY 965125671] UID validity status
* OK [UIDNEXT 5] Predicted next UID
* FLAGS (\Answered \Flagged \Deleted \Draft \Seen)
* OK [PERMANENTFLAGS (\* \Answered \Flagged \Deleted
```

```
Ä\Draft \Seen)] Permanent flags
* OK [UNSEEN 1] first unseen message in /var/spool/mail/craig
a0002 OK [READ-WRITE] SELECT completed
a0003 fetch 1 body[text]
* 1 FETCH (BODY[TEXT] {1440}
... an e-mail message that is 1440 bytes long ...
* 1 FETCH (FLAGS (\Seen))
a0003 OK FETCH completed
a0004 store 1 +flags \deleted
* 1 FETCH (FLAGS (\Seen \Deleted))
a0004 OK STORE completed
a0005 close
a0005 OK CLOSE completed
a0006 logout
* BYE ani.foobirds.org IMAP4rev1 server terminating connection
a0006 OK LOGOUT completed
Connection closed by foreign host.
```

Again, the first three lines and the last line come from telnet; all other messages come from IMAP. The first IMAP command entered by the user is LOGIN, which provides the username and password from /etc/passwd used to authenticate this user. Notice that the command is preceded by the string a0001. This is a *tag*, which is a unique identifier generated by the client for each command. Every command must start with a tag. When you manually type in commands for a test, you are the source of the tags.

IMAP is a mailbox–oriented protocol. The SELECT command is used to select the mailbox that will be used. In Listing 11.2, the user selects a mailbox named *inbox*. The IMAP server displays the status of the mailbox, which contains three messages. Associated with each message are a number of flags. The flags are used to manage the messages in the mailbox by marking them as Seen, Unseen, Deleted, and so on.

The FETCH command is used to download a message from the mailbox. In Listing 11.2, the user downloads the text of the message, which is what you normally see when reading a message. It is possible, however, to download only the headers or flags.

In this example, after the message is downloaded, it is deleted. This is done by writing the Deleted flag with the STORE command. The DELETE command is not used to delete messages; it deletes entire mailboxes. Individual messages are marked for deletion by setting the Deleted flag. Messages with the Deleted flag set are not deleted until either the EXPUNGE command is issued or the mailbox is explicitly closed with the CLOSE command, as is done in Listing 11.2. The session in Listing 11.2 is then terminated with the LOGOUT command.

Clearly, the IMAP protocol is much more complex than POP. It is just about at the limits of what can reasonably be typed in manually. Of course, you don't really enter these commands manually. The desktop system and the server exchange them automatically. They are shown here only to give you a sense of the IMAP protocol. About the only IMAP test you would ever do manually is to test whether imapd is up and running. To do that, you don't even need to log in. If the server answers the telnet, you know it is up and running. All you then need to do is send the LOGOUT command to gracefully close the connection.

Running the POP and IMAP Daemons

The test in Listing 11.1 shows POP running, and the test in Listing 11.2 shows imapd up and running. However, a test on a freshly installed Red Hat system returns the Connection refused error.

```
$ telnet localhost imap
Trying 127.0.0.1...
telnet: connect to address 127.0.0.1: Connection refused
$ telnet localhost pop3
Trying 127.0.0.1...
telnet: connect to address 127.0.0.1: Connection refused
```

Among the possible causes for this error on the localhost may be that you have not installed POP or IMAP. POP and IMAP can be installed during the initial installation, or installed later using RPM. If your system does not have POP and IMAP, the source code for a popular version of these daemons can be obtained via anonymous FTP from ftp.cac.washington.edu, where it is stored in the /imap/imap.tar.Z file. Our sample Red Hat system does have POP and IMAP installed, as Figure 11.1 illustrates.

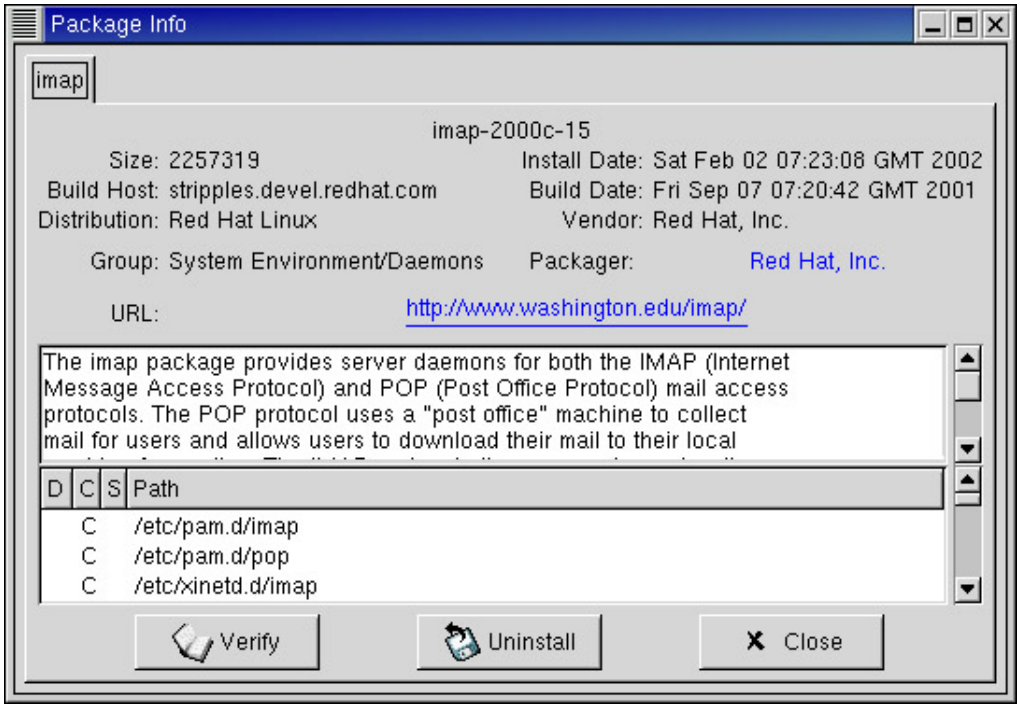


Figure 11.1: RPM query of the IMAP package

Both POP and IMAP are contained in the IMAP RPM. Figure 11.1 shows the gnorpm query of the IMAP package on our sample Red Hat system. The query shows that IMAP is installed, and examining the details provided by the query would show that POP is included in the package. Yet, neither the IMAP daemon nor the POP daemon responds to connections. The Connection refused error appears even on our sample Red Hat system, on which the software was installed during the initial installation.

Red Hat starts both POP and IMAP from xinetd. The /etc/xinetd.d directory contains five different xinetd configuration files relating to POP and IMAP:

imap xinetd uses this file to start IMAP when traffic arrives on port 143.

ipop2 xinetd uses this file to start POP2 when traffic arrives on port 109. POP2 is rarely used.

ipop3 xinetd uses this file to start POP3 when traffic arrives on port 110. POP3 is the version of POP most often used.

imaps xinetd uses this file to start IMAP version 4 when traffic arrives on port 993. Port 993 is used for IMAP traffic when that traffic is encapsulated in an SSL tunnel.

pop3s xinetd uses this file to start POP3 when traffic arrives on port 995. Port 995 is used for POP traffic when that traffic is encapsulated in an SSL tunnel.

All of these services are disabled by default on a Red Hat system. Use chkconfig to enable the services you desire. For example, to enable IMAP, you might enter the following:

```
[root]# chkconfig --list imap
imap            off
[root]# chkconfig imap on
[root]# chkconfig --list imap
imap            on
```

Of course, not all systems use xinetd. Some start POP and IMAP using inetd. If your system uses inetd and the service does not start on demand, it is probably commented–out of the inetd.conf file.

```
$ grep imapd /etc/inetd.conf
#imap  stream  tcp  nowait  root  /usr/sbin/tcpd imapd
```

Remove the # at the beginning of the imapd entry in the inetd.conf file to enable IMAP, and then send the inetd process a SIGHUP signal to cause it to re–read the configuration.

Rerunning the telnet test after editing inetd.conf or enabling IMAP with the chkconfig command should show that IMAP is running on your server because neither the POP daemon nor the IMAP daemon requires any special configuration. All users who have a valid user account on the system are allowed to download mail via POP or IMAP.

Using POP or IMAP from a Client

You're responsible for giving the user the correct information to configure his mail agent. The user needs to know the following:

- The hostname of the mail server
- The username and password required by the mail server
- Whether POP or IMAP should be used

Figure 11.2 shows a user configuring this information for the Netscape Communicator.

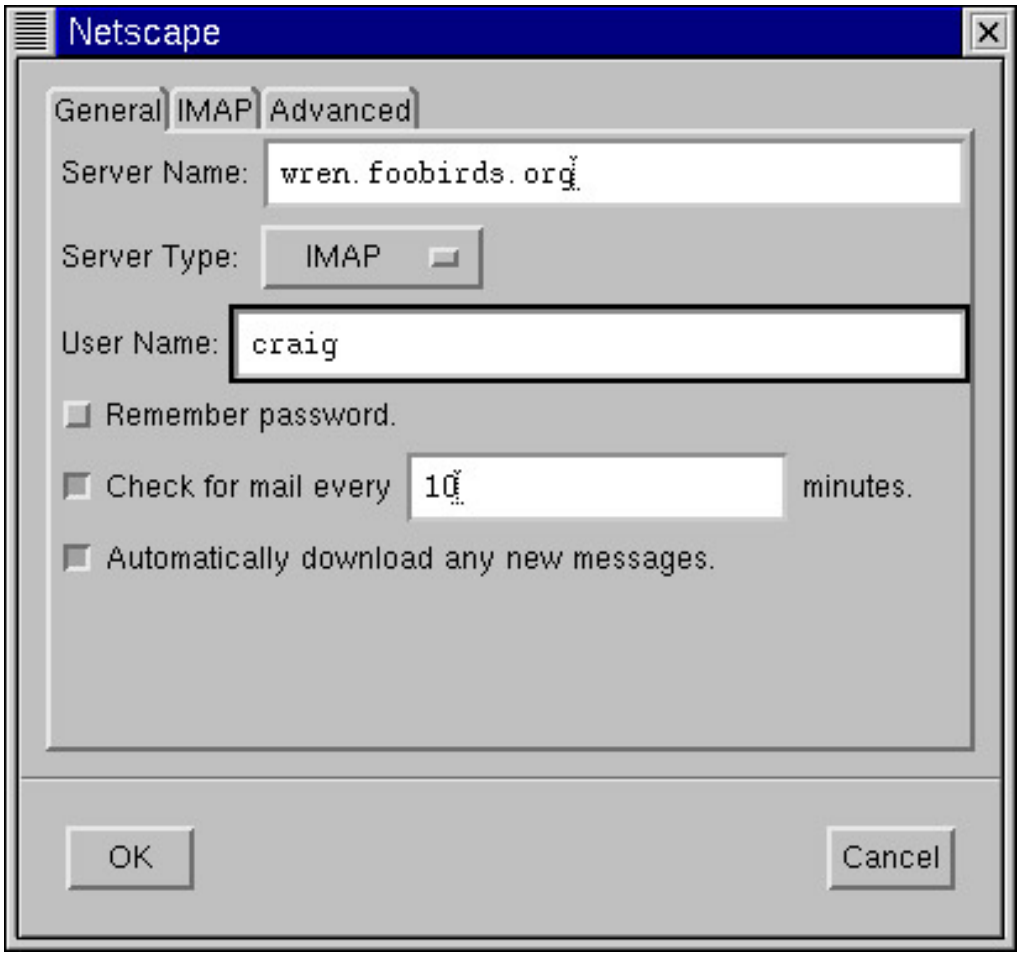


Figure 11.2: Configuring the mail client

This particular window permits the user to select POP3, IMAP, or something called *Movemail* from the Server Type list. By now, the meaning of POP3 and IMAP is obvious, but Movemail is something new. Movemail simply copies mail from the system's mail spool directory to the user's Netscape mail directory. Movemail works only when the local computer is also the mail server. Of course, with a Linux computer, this is possible.

Notice also that the configuration window allows you to name two different computers for outbound mail and inbound mail. It is possible for the departmental mailbox server and the server that forwards mail to the outside world to be two different computers.

Stopping Spam E–Mail

SPAM is a world–famous canned luncheon meat from Hormel Food. *Internet spam* isn't—it's junk e–mail. Spam e–mails are the unsolicited advertisements you receive that try to sell you a college diploma, pheromones that women can't resist, or pornography if those pheromones don't work. I'm sure you know what I mean because everyone attached to the Internet gets tons of this stuff.

One of your tasks as the administrator of a mail server is to reduce the amount of junk mail moving through the network. The techniques used for that task are the topic of this section.

Spam, Spam, Spam, Spam, and Spam

In a classic Monty Python comedy skit, John Cleese plays a waiter reciting a menu. In the beginning, the menu has one SPAM selection, but with each recitation of the menu, more and more of the items become SPAM, until he finally describes the menu as "Spam, spam, spam, spam, and spam."

Similarly, e–mail spam replicates itself through every possible mailing list until you find yourself with a mailbox full of exactly the same message repeated over and over again. It is this mindless repetition that gives spam e–mail its name.

Don't Be a Spam Source

Your first duty in the spam war is to make sure that your system is not a source of spam. Your system can be a source of locally generated spam, or it can be a relay for spam generated elsewhere. You need to respond to both possibilities.

Define an Acceptable Use Policy

To prevent locally generated spam, you need to make sure that everyone using your server knows that sending unsolicited advertisements from your server is not allowed. I have to admit that the idea of such a thing is alien to me. The government agencies and large businesses that I work with would fire *anyone* who misused corporate property in any way, let alone anyone running a private advertising firm on a corporate server!

But your situation may be different. You may be offering a community service on your system. In that case, you need a written Acceptable Use Policy (AUP) that tells people what type of use is allowed and what isn't. If you're not sure what an AUP should look like, ask your ISP, or check a national ISP. All large ISPs have some form of AUP.

Run the Identification Daemon

Running the auth server (identd) also helps to discourage homegrown spammers. The identification daemon monitors port 113. If it gets a request from a remote system, it tells that system the name of the user running the current connection process to that system. This allows remote mail servers to put a real username on the Received: header in incoming e–mail.

Some security experts shy away from identd because it sends local usernames to remote systems. However, hiding usernames does not provide any real security, so using identd to provide the name of the user running a process is a minimal risk. Red Hat even includes an encrypting version of identd, called pidentd, with a distribution that further minimizes the risk. pidentd encrypts outgoing identd responses with a secret key. In normal usage, this ends up being just gibberish to the remote site, which ensures that remote sites can't use the identd response to harvest username. But if a security incident occurs, the remote system administrator can come to you, and after you determine that the request is legitimate, you can decrypt the string and track down the problem user.

Properly Configure Mail Relaying

In addition to discouraging local users from generating spam, you need to discourage remote users from using your server as a tool for distributing spam. Nobody likes spammers, and the spammers know it. They do their best to hide the true source of the spam by relaying their junk mail through other people's servers. If your mail server allows relaying, spammers can make use of it.

To discourage spam, the default configuration of sendmail properly handles local mail, but does not relay messages for any outside sources. This is just the opposite of versions of sendmail before release 8.9, which relayed all mail by default. If your system runs an older version of sendmail, you should upgrade to get the full range of anti–spam tools.

Blocking all relaying works in most cases because most systems that run sendmail are not mail servers—they're desktop Linux and Unix systems dedicated to a single user. Because the user's mail originates on the system that is running sendmail, the mail is handled as local mail, and relaying is not required.

Blocking all relaying doesn't work if the system is a mail server. Most of the mail that a mail server delivers originates on its clients—these might be Microsoft Windows PCs that don't run their own sendmail program. Blocking relaying at the server causes the client to get an error when trying to deliver mail.

To create a mail server, you must allow some level of relaying. Select the correct feature from the following list of sendmail features to relax the relay restrictions just enough to get the job done:

FEATURE(`promiscuous_relay') Tells sendmail to relay mail from all sources.

FEATURE(`relay_entire_domain') Tells sendmail to relay mail from any local domain; that is, any domain defined in class M. (Don't remember the sendmail classes? Refer to Chapter 5.)

FEATURE(`relay_based_on_MX') Tells sendmail to relay mail for any host for which the local host is the MX server.

FEATURE(`relay_local_from') Tells sendmail to relay mail that contains the local domain in the MAIL FROM: header.

FEATURE(`accept_unresolvable_domains') Tells sendmail to accept mail from a host, even if it cannot be found in DNS or the host table. Normally, mail from hosts that do not exist in the domain name system is rejected.

To turn a system that blocks all relaying into a mail server, create a configuration file that allows the appropriate level of relaying. For example, you could create a variation on the foobirds.m4 DOMAIN file used in Chapter 5. Listing 11.3 shows such a variation.

Listing 11.3: Permitting Mail Relaying

```
divert(0)
VERSIONID(`foobirds.m4 03/16/2002')
define(`confFORWARD_PATH', `$z/.forward.$w+$h:$z/.forward+$h:$z/
Ä.forward.$w:$z/.forward')dnl
define(`confMAX_HEADERS_LENGTH', `32768')dnl
FEATURE(`relay_entire_domain')
FEATURE(`redirect')dnl
FEATURE(`use_cw_file')dnl
EXPOSED_USER(`root')
MASQUERADE_AS(foobirds.org)
FEATURE(masquerade_envelope)
FEATURE(genericstable)
```

This file contains all of the same features described in Chapter 5, plus the relay_entire_domain feature. This additional feature permits you to use the M class as a way to identify those hosts whose mail the server should relay.

Warning Be careful that you don't weaken the configuration so much that you become a spam source! All of the features listed above weaken the barrier to mail relaying, but some are worse than others. promiscuous_relay should not be used because it turns the system into a potential spam relay. Avoid the relay_local_from feature because it is very easy for spammers to write anything they want in the MAIL FROM: header, including your local domain name. Additionally, accept_unresolvable_domains should not be used unless it is absolutely required. It is intended for when sendmail really can't resolve domain names, such as on a laptop Linux system that does not always have access to a DNS server.

Using *sendmail* to Block Spam

The world will be grateful that your server is not a source for junk mail, but your users will be happy only if they are not the targets for spam. Two techniques that sendmail provides for blocking incoming spam are a DNS–based service to block spam sources, and a local database that controls access. This section examines both techniques.

Using the Realtime Blackhole List

The simplest way to block spam is to let someone else do it. sendmail allows you to use the Realtime Blockhole List (RBL) that comes from the Mail Abuse Prevention System (MAPS). Visit the website at mail–abuse.org/rbl to find out more about the MAPS system.

Using the RBL is very easy because the system is implemented through DNS. Every Linux system can issue DNS queries, so this is a very effective way to distribute information. Of course, a program can make use of the information only if it understands it. sendmail does. If you want to use the MAPS RBL to block spam, add the following feature to your sendmail configuration:

```
FEATURE(`dnsbl')
```

With this feature enabled, mail from every site listed in the MAPS RBL is rejected.

There are many systems listed in the MAPS RBL. MAPS enforces a very stern policy. Any site that relays spam—which could be your site if you don't properly configure sendmail—is listed in the RBL. The RBL is one of the reasons why it is essential to configure relaying properly. A mistake in configuring relaying could get your site added to the blackhole list. If a site stops relaying spam, it is removed from the list after about a month. If your site gets added to the RBL, apply to have it removed from the list by following the instructions on the MAPS website.

Although this is simple, it isn't perfect because you can't choose which sites listed in the RBL are rejected. It is an all–or–nothing proposition. In fact, that's what makes it as easy as turning a light switch on or off. This means that you might be blocked from receiving e–mail from a friendly site just because the administrator at that site forgot to turn off relaying. For this reason, some organizations decide to build their own DNS–based blackhole list.

The dnsbl feature accepts two arguments. The first of these is the name of the domain that contains

the blackhole list, which defaults to the home of the MAPS RBL. Point this argument to the domain in which you build your own blackhole list. If the DNS administrator created such a list in the dnsbl.foobirds.org domain, the following command would configure sendmail to make use of it:

```
FEATURE(`dnsbl', `dnsbl.foobirds.org')
```

For this to work, the DNS administrator needs to place the proper entries in the DNS server. Blackhole entries are simply DNS address records. The records are constructed by reversing the IP address of the blacklisted system to create a DNS name field for the record and using the address 127.0.0.2 as the data field of the address record. This format means that hosts are blacklisted by IP address instead of by name. For example, to blacklist clueless.nexploited.com, whose IP address is 192.168.72.37, place an address record for the hostname 37.72.168.192 within your blackhole domain. Some sample blackhole entries from the dnsbl.foobirds.org zone file might look like the following:

```
18.12.20.172.dnsbl.foobirds.org.     IN A 127.0.0.2
16.16.31.172.dnsbl.foobirds.org.     IN A 127.0.0.2
37.72.168.192.dnsbl.foobirds.org.    IN A 127.0.0.2
9.200.168.192.dnsbl.foobirds.org.    IN A 127.0.0.2
```

After the DNS zone has been created and a sendmail configuration has been built with the correct dnsbl feature, the new configuration can be tested using the –bt option with the sendmail command. Listing 11.4 shows a test of the dnsbl.foobirds.org domain.

Listing 11.4: Testing the *dnsbl* Feature

```
$ sendmail -bt -Ctest-dnsbl.cf
ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
> .D{client_addr}172.16.12.1
> Basic_check_relay <>
rewrite: ruleset 192 input: < >
rewrite: ruleset 192 returns: OK
> .D{client_addr}192.168.72.37
> Basic_check_relay <>
rewrite: ruleset 192 input: < >
rewrite: ruleset 192 returns: $# error $@ 5 . 7 . 1
         $: "550 Mail from " 192 . 168 . 72 . 37
         " refused by blackhole site dnsbl.foobirds.org"
> ^D
```

The client_addr variable normally contains the IP address of the remote system that initiated the mail connection. Because this is a –bt test, there is no mail connection involved, so we use the .D command to place a test value in the client_addr variable. Next, the ruleset Basic_check_relay is run without any input address, which is why the address focus characters < and > are used, but do not enclose any address value. The Basic_check_relay ruleset processes the address found in client_addr. The first test shows that the local address 172.16.12.1 is an acceptable source for e–mail. Next, the address 192.168.72.37 is stored in client_addr, and ruleset Basic_check_relay is run again. This time, the address is found in the dnsbl.foobirds.org domain, and the Basic_check_relay ruleset returns the error message 550 Mail from 192.168.72.37 refused by blackhole site dnsbl.foobirds.org.

The second argument available for the dnsbl feature is the error message that should be displayed when mail is rejected because of the blackhole server. The default message is the one displayed at

the end of Listing 11.4. Its format is 550 Mail from $&{client_addr} refused by blackhole site *dnsbl–domain,* where $&{client_addr} is the IP address that was rejected and *dnsbl–domain* is the value from the first argument provided to the dnsbl feature. Although I see no advantage to changing the standard error message, you could change it to "Mail rejected. "$&{client_addr}" is a suspected spam relay." with the following command:

```
FEATURE(`dnsbl', `dnsbl.foobirds.org', `"Mail rejected.
Ä"$&{client_addr}" is a suspected spam relay."')dnl
```

As usual, the choice between using the MAPS RBL or building your own blackhole server is a choice between simplicity and flexibility. If your site is small, you might have a limited number of other sites with which you exchange mail, and you might believe that the likelihood of any of those sites appearing in the RBL is very slim. Additionally, you probably don't have the available staff necessary to build and maintain your own blackhole site. For these reasons a small site might choose simplicity. If you run the mail server for a large site, you may want to define your own e–mail access list to ensure that continued connectivity to all of the sites you want to reach is under your direct control. Creating your own blackhole server is one technique that sendmail provides for you to do that. Another is to define access controls in the access database.

Understanding the Access Database

The sendmail access database defines e–mail sources using e–mail addresses, domain names, and IP network numbers, along with the action that sendmail should take when it receives mail from the specified source. For example:

```
spammer@bigisp.com       REJECT
wespamu.com              REJECT
172.18                   REJECT
```

This database tells sendmail to reject any mail from the e–mail address spammer@bigisp.com, from any host in the domain wespamu.com, and from any computer whose IP address begins with network number 172.18. Each entry in the database begins with the source of the mail, followed by a keyword that tells sendmail what action to take. Table 11.3 lists the valid keywords and the actions they cause.

Table 11.3: Access Database Actions

Keyword	Action
DISCARD	Drops any message received from the specified source.
OK	Absolutely accepts messages from the specified source.
REJECT	Issues an error message, and drops any mail from or to the specified address.
RELAY	Relays mail coming from or bound to the specified address.
Error message	Returns the specified error message to the source address.

An extension to the database shown previously illustrates how these actions are used. Listing 11.5 contains this more–complete access database.

Listing 11.5: A Sample Access Database for *sendmail*

```
spammer@bigisp.com    REJECT
wespamu.com           REJECT
```

```
172.18                  DISCARD
example.org             OK
129.6                   RELAY
weselljunk.com          550 Junk mail is not accepted
```

The REJECT commands cause sendmail to return an error message to the source and then discard the mail. The DISCARD command drops the mail without sending any message back to the source. Most anti–spam authorities discourage silently discarding mail because they feel it does not discourage the spammer. For all he knows, you received the mail, so he just keeps sending more junk.

The OK command causes sendmail to accept mail from example.org, regardless of other conditions. For example, if mail arrives from a hostname that includes the example.org domain and cannot be resolved by DNS, sendmail accepts that mail, even though the accept_unresolvable_domains feature has not been enabled. To do this, you must, of course, fully trust example.org.

The RELAY command causes sendmail to relay mail for network 129.6, even though basic relaying is not enabled on the system. Like the OK command, using the RELAY command means that you fully trust every host on network 129.6. If you don't have anything else in your database, you probably want a RELAY entry like this one for your own network. As discussed earlier, sendmail blocks all mail relaying—even mail from your clients. An access database entry such as 172.16.5 RELAY enables relaying for every host attached to the specified network.

Using the Access Database in *sendmail*

After building the database, you also need to let sendmail know that you have an access database, and you want to use it. Use the access_db feature to do that.

Assume that you're using the configuration that was created in Chapter 5.We created two customized files: a DOMAIN file specifically for the foobirds.org domain and a linux.mc file to include the custom DOMAIN file in the sendmail configuration. Because the access database is specific to our server, let's add the necessary feature to the foobirds.m4 DOMAIN file, as shown in Listing 11.6.

Listing 11.6: Adding the Access Database to the Configuration

```
divert(0)
VERSIONID(`foobirds.m4 03/16/2002')
define(`confFORWARD_PATH', `$z/.forward.$w+$h:$z/.forward+$h:$z/
Ä.forward.$w:$z/.forward')dnl
define(`confMAX_HEADERS_LENGTH', `32768')dnl
FEATURE(`access_db', `hash -o /etc/mail/access')
FEATURE(`redirect')dnl
FEATURE(`use_cw_file')dnl
EXPOSED_USER(`root')
MASQUERADE_AS(foobirds.org)
FEATURE(masquerade_envelope)
FEATURE(genericstable)
```

This is the same foobirds.m4 file described in Chapter 5, with one addition. The first FEATURE listed in this file invokes the access database and describes where it is located. As described in Chapter 5, the foobirds.m4 macro file is referenced in the macro control file linux.mc. Process the control file with the m4 command to produce a sendmail.cf file that uses the new database:

```
# m4 ../m4/cf.m4 linux.mc > sendmail.cf
```

After you create and install the new sendmail.cf file with m4, your new access database is in force and is blocking spammers. If you need even more control over the process, you can define your own anti–spam sendmail rewrite rules.

Using Anti–Spam Rewrite Rules

Most administrators think of sendmail rewrite rules as a way to modify addresses on outbound e–mail that originates on the local system in the user's mailer. The anti–spam rulesets allow you to process the addresses and headers from incoming mail. sendmail provides three anti–spam rulesets specifically for your personal rules:

Local_check_relay A ruleset in which you can define rules for handling mail that is being relayed

Local_check_rcpt A ruleset in which you can define rules to process inbound mail based on the recipient address

Local_check_mail A ruleset in which you can define rules to process inbound mail, based on the sender address

Suppose that you have been receiving junk mail that is trying to masquerade as local mail by using a From address that contains only a username. Further, assume that you have configured your mail server so that the From address of local mail always includes the hostname. You could use Local_check_mail to check the sender address, as shown in Listing 11.7.

Listing 11.7: A *Local_check_mail* Example

```
SLocal_check_mail
# Check for user@host
R$+@$+   $@$#OK
R$*      $#error $: 550 Invalid From address
```

The first line in this example is an S command that defines the ruleset named Local_check_ mail. The first R command matches the incoming address against the pattern $+ @ $+, which looks for one or more tokens ($+), a literal at sign (@), and one or more tokens. Any address in the form of *user@host* matches this pattern. The transformation says that if the address matches the pattern, exit the ruleset ($@) and return the mailer name $#OK to the calling ruleset. ($#OK is a phony mailer used to indicate that the address is valid.)

The second R command matches every address that failed to match the first rule. For all of these addresses, the rule returns the mailer name $#error and the text of an error message. The $#error mailer is a special mailer that returns the mail to the sender along with an error message. An alternative to this would be the $#discard mailer, which silently discards the mail. Most administrators prefer to return an error message.

In addition to these rulesets, you can call a ruleset from a header definition to check the format of the headers your system receives. Sometimes spammers use malformed headers that indicate the mail is spam. Suppose that you get spammed by someone who forgets to create a valid–looking Message–ID header. You could use code such as the code shown in Listing 11.8.

Listing 11.8: An Example of Creating a Local Ruleset

```
LOCAL_RULESETS
HMessage-Id: $>check_MID_header

Scheck_MID_header
R$+@$+            $@ $#OK
R$*               $#error $: 550 Invalid Header
```

The LOCAL_RULESETS section contains an H command for the sendmail.cf file. Unlike the H commands shown in Chapter 5, this one doesn't contain a header format. Instead, it uses the $> syntax to call a ruleset to process the header. This example calls a ruleset named check_MID_header because that is the name of the new ruleset defined in Listing 11.8.

The Scheck_MID_header command is the first line of ruleset check_MID_header. This ruleset is essentially identical to the one described in the previous example. It checks to make sure that the Message−ID header contains both a unique message identifier and a hostname in the form *identifier@host.* All other formats are rejected as errors.

These rewrite rules are simply examples created to illustrate the way local rulesets are defined and used. They are not applicable to a real configuration. Frankly, developing rewrite rules to fight spam is not widely recommended. First, rewrite rules can be complex and difficult to develop, making the cure worse than the disease. Second, the format of spam mail is constantly changing, making the rule written today useless tomorrow. Most administrators find it better to rely on the blackhole list, the access database, and the ability of the user's mailer to filter mail.

Filtering Out Spam at the Mailer

Despite your best efforts, spam and other unwanted mail *will* get to your users. This is partly because you can't block all of the spam, and partly because not all unwanted e−mail is spam. Sometimes, a user just doesn't want to look at some legitimate e−mail simply because of personal preference. In this case, the mail needs to be filtered at the user's mail reader. Most mail readers provide this capability. This section provides a few examples.

Using the *elm* Filter

elm is an old terminal mode mailer. elm has some die−hard fans, but most users have moved on to mailers that include a graphical user interface. However, the elm source code distribution comes with a filtering tool aptly named filter. The source code is available from ftp.virginia.edu, where it is stored in the /pub/elm directory. Even if you don't use elm, you can use the filter program to process incoming mail.

The filter program is invoked with the .forward file (covered in Chapter 5 during the discussion of sendmail aliases). Its primary purpose is to provide each user with a way to specify personal mail forwarding. One feature of .forward is that it can forward mail to a program, which is the feature used to send mail to the filter program.

To process mail through the filter program, put the following line in .forward:

```
| "exec filter -o ~/filter.errors"
```

This assumes that you downloaded the filter source code, compiled it, and installed the executable

in the smrsh execution directory. Most Linux systems, Red Hat included, use smrsh as the prog mailer. When mail is forwarded to a program, as it is in this example, it is handled by the prog mailer (which, in this case, is smrsh). smrsh will execute only programs found in its execution directory. In our sample Red Hat system, that directory is /etc/smrsh.

The filter program reads its configuration from the filter−rules file in the .elm directory in the user's home directory. Therefore, if the home directory of the user running filter is /home/sara, filter looks for a file named /home/sara/.elm/filter−rules. If the user doesn't really run elm, she needs to create an .elm directory to hold the configuration file.

The filter−rules file contains the directions for filtering the mail. It is written as a series of "if" statements. If the incoming mail matches the condition defined by the statement, it is processed in the manner that the statement directs. A sample filter−rules file illustrates how this works:

```
if (from = "*.gov") then Save ~/Mail/clients
if (from = "neil@sybex.com") then Save ~/Mail/editors
if (from = "kylie@sybex.com") then Save ~/Mail/editors
if (from = "*.wespamu.com") then Delete
```

This set of filters routes the mail received from different sources to different mailboxes. Mail from government consulting clients is routed to the clients mailbox; mail from Sybex editors is routed to the editors mailbox. The first three rules route mail to different mailboxes. The last rule discards unwanted mail; everything from wespamu.com is deleted.

All of the lines in the filter−rules file have the same basic format:

```
if (condition) then action
```

The *condition* can test the contents of the From:, To:, Subject:, and Sender: headers. The content string, which is enclosed in quotes, can be a partial string and can use wildcard characters to match more than a single case (*.gov is an example). The *action* can delete the message, save it in a mail folder, forward it to another address, or pass it to another program for further processing.

filter is easy for users to understand and use. However, filter is not delivered with all Linux systems because elm is no longer widely used. Most users tend to use the filtering mechanism that comes with the mail reader they use.

Filtering with Netscape

Many users use Netscape to read their e−mail. They can, of course, use the filter program or procmail (which is discussed next) to filter mail, even if they then read the mail with the Netscape mail reader. However, Netscape provides its own mail−filter capability, which is particularly suited to those users who prefer a graphical interface.

Note For these examples, we use the version of Netscape delivered with Red Hat 7.2.

From the Edit menu in the Netscape Messenger window, select Message Filters and then click on New in the Message Filters window to open the Filter Rules window shown in Figure 11.3.

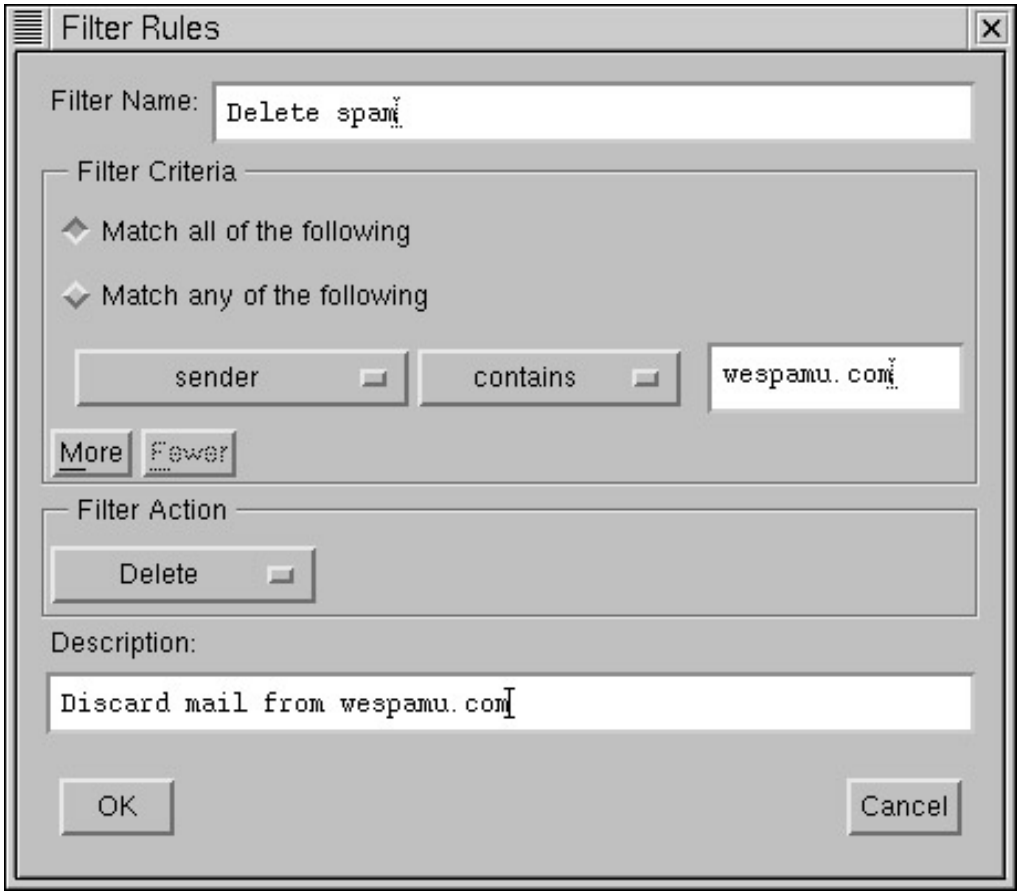


Figure 11.3: Defining Netscape filter rules

In many ways, these rules are very similar to those used in the filter program. Again, the user constructs if/then statements, but this time, a graphics template is used to do it. Figure 11.3 shows the same rule that was applied to wespamu.com in the filter example: If the sender name contains the string wespamu.com, then delete the message.

The first drop–down list in the Filter Criteria section of the Filter Rules dialog box defines the item in the mail that is being tested. In Figure 11.3, sender is selected, which filters based on the sender's e–mail address. The drop–down list also allows you to match on the following:

subject Filters mail based on the content of the Subject: header.

body Filters mail based on the content of the message body.

date Filters mail based on the date the mail was created. Selecting this value changes the list of available conditional tests to is, isn't, is before, and is after.

Priority Filters mail based on its priority. Selecting this item changes the list of available conditional tests to is, isn't, is higher than, and is lower than. It also causes a new drop–down box to appear that contains the possible priority selections. These are lowest, low, normal, high, and highest.

status Filters mail based on whether or not it has been read or replied to. Selecting this item changes the list of available conditional tests to is and isn't, and causes a new drop–down box to appear that contains the selections read and replied.

to Filters mail based on the e–mail addresses in any of the To: headers.

CC Filters mail based on the e−mail addresses in any of CC: headers.

to or CC Filters mail based on any recipient addresses. Every e−mail address in every To: or CC: header is checked.

age in days Filters mail based on how old it is in days. Selecting this value changes the list of available conditional tests to is, is greater than, and is less than.

In addition to all of these standard matches, you can select the Customize Headers item from the end of the first drop−down box to specify any header that you want to match against. The Customize Header selection opens a box from which you choose a previously defined header, or you can click the New button to enter a header name. Any valid header name can be used. If the header is encountered in the incoming mail, its content is matched against the filter.

The second drop−down box defines the conditional test. In Figure 11.3, contains is selected. This means the filter checks to see if the sender address contains the string defined in the next box, which is wesapmu.com in the figure. Depending on the conditional test selected, the value that is tested against can be a full or partial value; contains permits a partial value to match. The other choices that are available when the sender address is being tested are as follows:

doesn't contain The test evaluates to true, and the action is taken if the tested field does not contain the specified value.

is The test evaluates to true, and the action is taken if the tested field is equal to the specified value.

isn't The test evaluates to true, and the action is taken if the tested field is not equal to the specified value.

begins with The test evaluates to true, and the action is taken if the string at the beginning of the tested field exactly matches the specified value.

ends with The test evaluates to true, and the action is taken if the string at the end of the tested field exactly matches the specified value.

This list of conditional tests is available for the sender address, the subject header, the to address, and the CC address. All conditional tests except begins with and ends with are available for testing the content of the message body, and all except is and isn't are unavailable when testing to or CC. As noted, when other values are tested, they generate their own list of valid conditionals.

The last drop−down box in the Filter Rules dialog box selects the filter action. In the example, the action is Delete. Other available actions are the following:

move to folder Mail that matches the filter is routed to a specific mailbox. If this action is selected, two additional items appear in the window. One is a drop−down list that allows you to select the mailbox to which the mail is sent. The other item, labeled New Folder, permits you to define a new mailbox to receive the mail.

change priority The priority of mail that matches the filter is changed to the priority you select. Selecting this action causes a drop−down box to appear that contains the possible priority selections. This box is identical to the one that appears if priority is selected as the item being tested in the first drop−down box.

mark read Mail that matches the filter is marked as read.

ignore thread Mail that matches the filter is part of a thread that is ignored. A *thread* is related mail or news postings on the same topic.

watch thread Mail that matches the filter is part of a thread that is being monitored.

A filter can contain multiple rules. Click the More button to add additional rules. Use the Match All Of The Following check box to filter mail only if it matches every rule defined for the filter, or use the Match Any Of The Following check box to filter mail if it matches any rule in the filter.

In addition to defining the rules, give each filter a name, and (optionally) give it a description. After it's defined to your satisfaction, click OK, and the rule is enabled.

Managing Mail with *procmail*

As mentioned in Chapter 5, procmail is the default local mail–delivery program for Linux systems. procmail provides the most powerful and complex e–mail filtering system available for Linux. procmail filters are defined by the user in the .procmailrc file. Additionally, the system administrator can define system–wide filters in the /etc/procmailrc file. The format of both files is the same. The system administrator uses the /etc/procmailrc file for general anti–spam filtering. The end user can then use .procmailrc to add filtering for personal preferences. The examples in this section show both anti–spam filters and personal preference filters.

The .procmailrc file contains two type of entries: environment variable assignments and mail–filtering rules, which procmail calls *recipes*. Environment variable assignments are straightforward, and look just like these assignments would in a shell initialization script such as .bashrc. For example, HOME=/home/craig is a valid environment variable assignment.

Note See the procmailrc manual page for the full listing of the more than 30 environment variables.

The real substance of a .procmailrc file is the recipes. The syntax of each recipe is:

:0 [*flags*] [:*lockfile*]

[* *condition*]

action

Every recipe begins with :0, which differentiates it from an assignment statement. The :0 is optionally followed by flags that change how the filter is processed. Table 11.4 lists all of the flags and their meanings.

Table 11.4: procmail Recipe Flags

Flag	Meaning
A	Execute this recipe if the preceding recipe evaluated to true.
a	This has the same meaning as the A flag, except that the preceding recipe must also have successfully completed execution.
B	Filter based on the message body.
b	Pass the body of the message on to the destination. This is the default.

c	Create a carbon copy of this mail.
D	Tests are case–sensitive. By default, case is ignored.
E	Execute this recipe if the preceding recipe was not executed.
e	Execute this recipe if the execution of the preceding recipe returned an error.
f	Pass the data through an external filter program.
H	Filter based on the message headers. This is the default.
h	Pass the message header on to the destination. This is the default.
i	Ignore write errors for this recipe.
r	Write the mail out as is without ensuring that it is properly formatted.
w	Wait for the external filter program to finish, and check its exit code.
W	This is the same as the w flag, except no error message is printed.

An optional lockfile can be identified to prevent multiple copies of procmail from writing to the same mailbox at the same time. This can happen on a busy system, causing some pretty strange–looking mail. The *lockfile* name is preceded by a colon. If the colon is used and no name is specified, a default name created from the mailbox name and the extension .lock is used.

The conditional test is optional. If no *condition* is provided, the recipe acts as if the *condition* is true, which means that the action is taken. If a *condition* is specified, it must begin with an asterisk (*). The *condition* is written as a regular expression. If the value defined by the regular expression is found in the mail, the *condition* evaluates to true, and the action is taken. To take an action when mail does *not* contain the specified value, put an exclamation point in front of the regular expression. Here are some examples of valid conditional tests:

```
* ^From.*neil@sybex.com
* !^Subject: Chapter
```

The first conditional checks to see if the mail contains a line that begins with (^) the literal string From, which is followed by any number of characters (.*) and the literal string neil@sybex.com. The second conditional matches all mail that does not (!) contain a line that begins with the string Subject: Chapter. If multiple conditions are defined for one recipe, each condition appears on a separate line.

Note To learn more about regular expressions, see *Mastering Regular Expressions: Powerful Techniques for Perl and Other Tools* by Jeffrey Friedl (O'Reilly, 1997).

Although there may be multiple conditions in a recipe, there can be only one action. The action can direct the mail to a file, forward it to another e–mail address, send it to a program, or define additional recipes to process the message. If the action is an additional recipe, it begins with :0. If the action directs the mail to an e–mail address, it begins with an exclamation point (!); if it directs it to a program, it begins with a vertical bar (|). If the action directs the mail to a file, just the name of the file is specified.

A Sample *.procmailrc* File Using the information described previously, you might create a .procmailrc file such as the one shown in Listing 11.9.

Listing 11.9: A sample *.procmailrc* file

```
MAILDIR=$HOME/mail

:0 c
```

```
backup

:0:
* ^From.*@sybex.com
editors

:0 c
* ^From.*rdenn
* ^Subject:.*NT
!robert@bobsnet.org
   :0 A
   ntbook

:0
* ^From.*@wespamu.com
/dev/null

:0 B
* .*pheromones
| awk -f spamscript > spam-suspects
```

This sample .procmailrc file begins with an environment variable assignments statement. The statement assigns a value to the variable MAILDIR, and it uses the value of the HOME variable. Thus, it illustrates both assigning a variable and using a variable. Frankly, the statement is there just to illustrate how variables are used. It was not really needed for this file.

The first recipe in the file is:

```
:0 c
backup
```

This recipe makes a carbon copy of the mail, and stores it in a mailbox named backup. This recipe came straight from the .procmailrc documentation, in which it is suggested as a way to ensure that no mail is lost when you're first debugging the .procmailrc file. After all of the recipes work as you want them to, remove this recipe from the file so that you don't continue to keep two copies of every piece of mail.

The second recipe is:

```
:0:
* ^From.*@sybex.com
editors
```

This recipe puts all the mail that contains a line that begins with (^) the literal From, any number of characters (.*), and the literal @sybex.com into the mailbox named editors. The most interesting thing in this recipe is the first line. Notice the :0: value. From the syntax, you know that the second colon precedes the name of the lockfile. In this case, no lock filename is provided, so the name defaults to editors.lock.

The third recipe is:

```
:0 c
* ^From.*rdenn
* ^Subject:.*NT
!robert@bobsnet.org
   :0 A
   ntbook
```

This recipe searches for mail that is from someone named rdenn and that has a subject of NT. A carbon copy is made of the mail, and it is sent to robert@bobsnet.org. The other copy of the mail is stored in the ntbook mailbox.

The fourth recipe is:

```
:0
* ^From.*@wespamu.com
/dev/null
```

This recipe shows how spam mail is deleted using procmail. All mail from wespamu.com is deleted by sending it to /dev/null, the null device.

The final recipe is:

```
:0 B
* .*pheromones
| gawk -f spamscript > spam-suspects
```

This recipe illustrates how mail is passed to an external program for processing. All messages that contain the word pheromones anywhere in the message body are passed to gawk for processing. In this example, gawk runs a program file named spamscript that extracts information from the mail, and stores it in a file named spam−suspects. You can imagine that the administrator of this system created an awk program named spamscript to extract all of the e−mail addresses from suspected spam.

This range of recipes illustrates the power and flexibility of procmail. Despite the obscure syntax of a .procmailrc file, it may be the best tool for filtering e−mail.

In Sum

This chapter concludes Part 3, "Departmental Server Configuration." In this part of the book, configuration servers, file sharing, printer sharing, and mailbox servers have all been covered. Add these to the login services, name services, web servers, and routing covered in Part 2, and you have a complete network server.

The final part of the book, "Maintaining a Healthy Server," examines those tasks that are necessary to keep that complete server running in tip−top shape. Part 4 begins with a chapter on security, which is particularly critical for a network server because connecting to a network greatly increases the security threats to your server.

Part IV: Maintaining a Healthy Server

Chapter List

Chapter 12: Security
Chapter 13: Troubleshooting

Part Overview

Featuring:

- Tracking the latest security problems and fixes
- Using the tcpd wrapper program to improve security
- Using xinetd security features
- Using the built–in firewall features of Linux
- Improving Linux password security
- Monitoring your server for security problems
- Analyzing network trouble reports
- Using the basic Linux troubleshooting tools
- Looking for configuration errors
- Testing routing
- Testing Domain Name Service
- Analyzing network traffic

Chapter 12: Security

Overview

Good security is good system administration. Security is a fundamental part of running a reliable network server. Undoubtedly, your server will be attacked and compromised by people on the network. Your job is to reduce the number of successful attacks, to limit the amount of damage done, and to quickly recover from the attack. This chapter will help you do your job.

This is a book about Linux network servers, so it focuses on network security threats. Despite this emphasis, you should remember that network security is only part of the overall security of your system:

- Physical security is required to protect the server hardware and to prevent unauthor–ized access to the system console.
- Filesystem security, which is described in Chapter 9, "File Sharing," is necessary to protect the data on the server.

For a broader view of security, see *Linux Security* by Ramón Hontañón (Sybex, 2001). The focus of this chapter on network security is not meant to downplay the importance of physical and filesystem security, but security threats originating from the network are a major source of Linux server problems. This chapter describes those threats, and tells you how to face them.

Understanding the Threats

Connecting your server to a network gives it access—and makes it vulnerable—to everyone on the network. The larger the network, the larger the threat. When you connect your system to a network, you should assess the security threat that the network connection creates. To make this assessment, you need to consider the potential harm to your organization from a successful security attack.

The impact of a security attack depends on what system and what information are compromised. The loss of a key server affects many users, whereas the loss of a desktop client may affect only one user. Likewise, unauthorized access to a file containing plans for the office party cannot be compared to unauthorized access to your corporate strategic plan. Although your efforts should be directed toward protecting things that are important, every system requires some level of protection. A break–in on a small, insignificant system can end up compromising your entire network.

The Basic Threats

There are three basic threats to the information stored on your network:

Threats to the secrecy of data These are the unauthorized disclosures of sensitive data that can be caused by setting the wrong file permissions, by having someone improperly gain root privileges, or by having the data stolen directly off the wire.

Threats to the integrity of data These are the unauthorized modifications of data that can be caused by using the wrong file permissions or by someone improperly gaining root privileges. This is a common threat to web servers where intruders change data in obvious and embarrassing ways. But a more insidious threat is the

possibility of subtle modifications to data that are designed to undermine the reputation of an organization. After a system has suffered an unauthorized access, all files on the system are suspect.

Threats to the availability of data These attacks deny legitimate access to the data. If files are improperly protected or an intruder gains root access, files can be deleted. Vandals can also launch a Denial of Service (DOS) attack to overwhelm your server, blocking access to your data when you need it.

The network threats that lead to these data problems are as follows:

Unauthorized access This is any time that someone who should not have access to your system is able to access it without permission.

Denial of Service Any attack that is designed not to gain access to your system but to prevent you from using your system.

All networked systems are vulnerable to these attacks. Luckily, Linux provides a range of tools to help you reduce the threat.

A Reality Check

Legend has it that network threats come from sophisticated code hackers who have a deep understanding of networks and operating systems. These legendary characters are motivated by espionage or a desire to force unresponsive computer corporations to improve their software. I wish it were true! If it were, these people would have no interest in attacking my Linux system.

Unfortunately, the reality is that most attacks come from unskilled people running canned attack scripts. The scripts have become so simple to use that the people who now use them are "script kiddies." The people who run these scripts are not interested in espionage, but they don't mind causing a little mayhem! Additionally, if they were truly rebels against the corporate system working to improve the security of operating system software, they would be writing new Linux code. After all, Linux is open source code; no one can claim that a corporate monolith is keeping the code hidden.

Given this, you might guess that Linux is not a target for security attacks. You would guess wrong. Unfortunately, Linux is one of the most popular targets for attack. A study conducted a few years ago by Peter Mell of NIST showed that attack scripts for Linux are as popular as scripts for Windows NT, and that these two systems were the most popular operating systems for attack scripts.

Clearly, open source code is no protection from attack. The people who run attack scripts are not motivated to "fix" the system—they are just looking for easy targets. Your job is to make sure that your system isn't an easy target.

Look at it this way. The bad news is that you don't have to be important to be a target of security attacks. The good news is that the guy at the other end of the attack isn't a network guru. If you can track the vulnerabilities exploited by the "script kiddies," and close those holes as they appear, your system will be reasonably secure.

Keeping Informed

To secure a system, you need to know its vulnerabilities. Your goal should be to stay as well–informed about Linux's vulnerabilities as the vandals are. Frankly, you won't be able to. *You* have a life and responsibilities, so the vandals who have nothing better to do will get ahead of you, and may compromise your system. Despite the difficulty, you should do your best to keep up–to–date about security problems.

There are several good sources of information about known security vulnerabilities:

- Your Linux vendor's website should have useful security information specific to your Linux distribution.
- General information about the bugs that create security vulnerabilities is available in the Bugtraq archive, which can be found on the web at http://www.securityfocus.com/.
- Security advisories are available from http://www.l0pht.com/, csrc.nist.gov, http://www.cert.org/, and other sites.
- A good site for Linux software updates and security hole announcements is http://www.freshmeat.com/.
- The SANS (System Administration, Networking and Security) Institute offers informative security newsletters that are delivered weekly via email. It also has a useful online reading room. These resources are available from its website http://www.sans.org/.

Track all of the problems that pertain to Linux, Unix, and Unix applications—all of these could affect your Linux server. Figure 12.1 shows an example of the Bugtraq archive at http://www.securityfocus.com/.

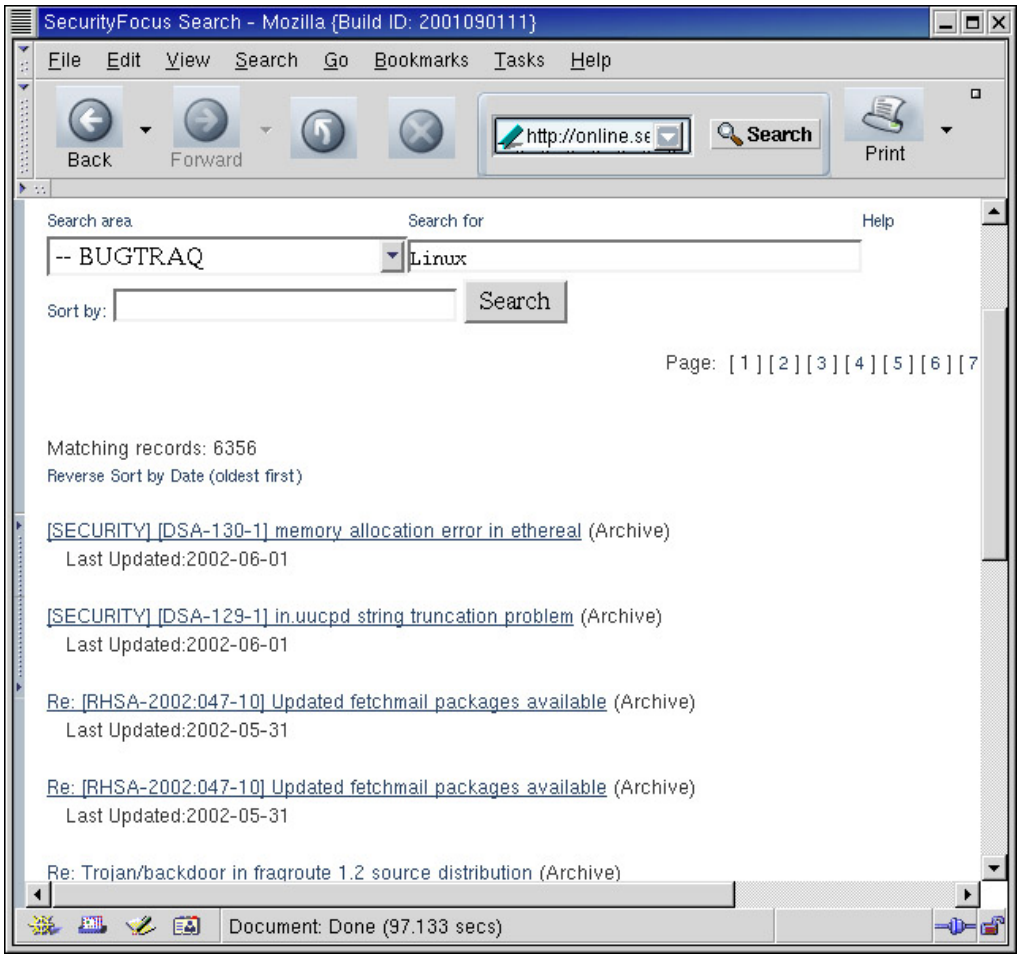


Figure 12.1: Searching the Bugtraq Archives

Figure 12.1 is the result of searching the Bugtraq Archives for the string Linux. Clicking a link takes you to the bug report, so you can read it and determine whether this bug is a threat to your system.

In addition to visiting the sites that report bug and security problems, visit the http://www.hackers.com/ website. It provides information about security exploits. The site gives you access to the same scripts that intruders use to attack your system. http://www.hackers.com/ gives a description of the exploit, the exploit technique, and the defense against the exploit. Use this information to make sure that your system is not vulnerable to the old attacks and to evaluate the new attacks as they appear in order to understand the vulnerabilities they exploit. In addition to providing descriptions of current exploits, this site gives information about what is currently going on in the network security world.

Figure 12.2 shows the Linux exploits report from http://www.hackers.com/ as it existed in June 2002. Clicking on View Exploit for any of the listed exploits takes you to a page that describes the exploit. From there, you can follow links to pages that tell you how to exploit the vulnerability and to pages that tell you how to defend against the exploit.

Figure 12.2: Linux exploits found at http://www.hackers.com/

Closing the Holes

Most intruders enter systems through well–known holes in the system software. The most important thing you can do to improve the security of your system is close the holes by installing security updates as soon as they become available.

Vulnerabilities are not limited to the Linux kernel itself. In fact, most of the vulnerabilities that are exploited occur in the network software that runs on your Linux system. In the first–quarter 2002 Top Ten Vulnerabilities list at http://www.securityfocus.com/, some of the top 10 vulnerabilities are pieces of user–level software that run on Linux systems. Clearly, it is not enough to keep the Linux operating system up–to–date. You must keep all software packages updated.

Finding the Latest Software

To update software, you need to need to know what software needs to be updated and where to find it. Security advisories (such as those found at SANS, NIST, and CERT) usually describe the problem and tell you the solution; often, they point you to the appropriate software fix. Even the bug reports found in the Bugtraq Archive sometimes include fixes, as mentioned in the discussion of Figure 12.1.

The vulnerability report shown in Figure 12.3 includes links to the software updates that fix the reported problem. Clicking a link retrieves the fix that can then be installed. Figure 12.3 shows a page from the ICAT security database at NIST.

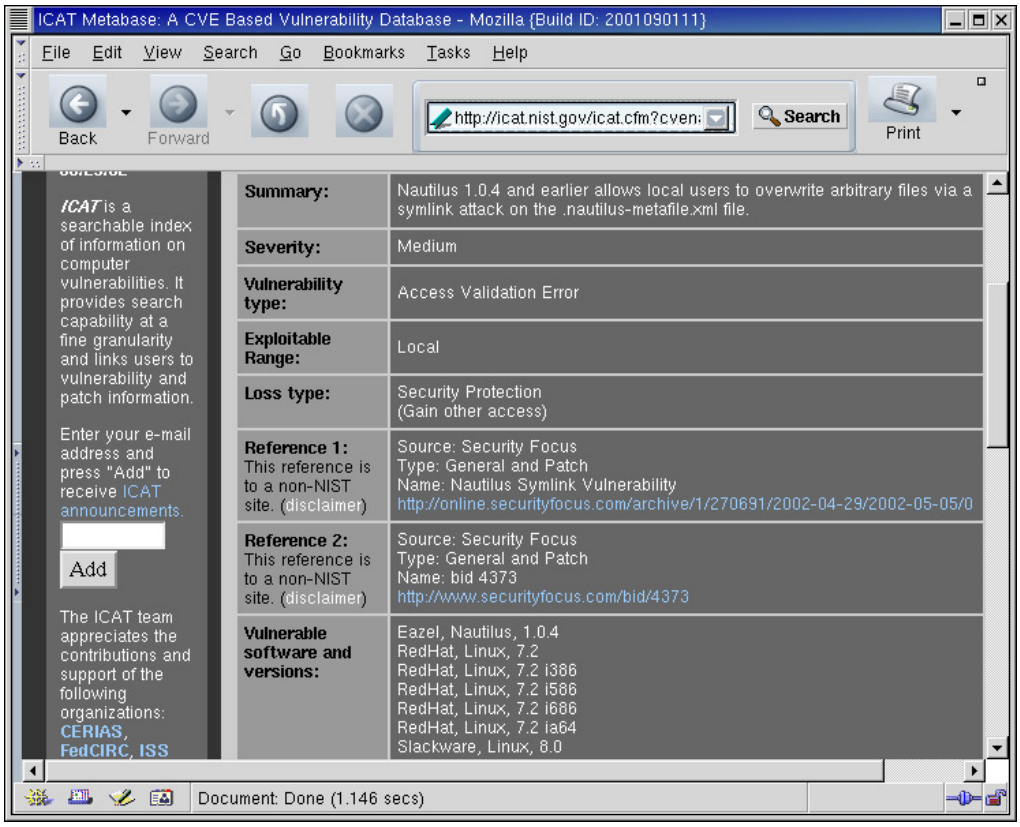


Figure 12.3: Locating software updates from a vulnerability report

Unfortunately, the fix is not always included in a vulnerability report, so you may need to look for it yourself. Go to your Linux vendor's website, and look for the latest security updates.

Figure 12.4 shows a security report list on the Red Hat website. Clicking on an item in the list takes you to the report. The report describes the problem, and provides a link to the software update. If you're looking for a specific fix, you can search through the advisories on the web page. The big advantage of this is that you will often find fixes for bugs that you have never heard of. On the downside, sometimes you find out that a bug you have heard of has not yet been fixed. Nevertheless, you should take all of the bug fixes that are offered, and periodically check back to see whether the bug you're concerned about has been fixed.

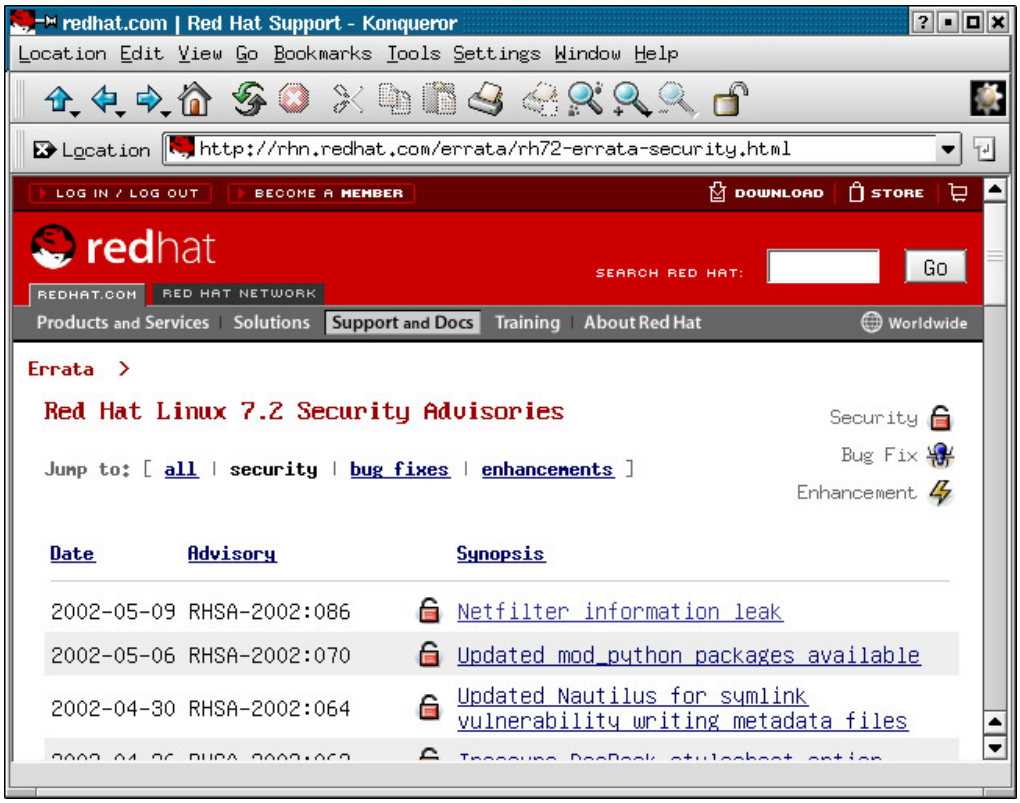


Figure 12.4: Red Hat provides security reports online.

Tip Frequently, administrators complain that the authors of software don't fix bugs, but a much more common problem is that system administrators don't use the bug fixes that are out there. Set aside a time each month to download and install the fixes provided by your software vendor. Make it part of your routine.

Removing Unneeded Software

Reduce the burden of keeping software updated by removing all of the software you don't really need. Traditionally, software packages such as sendmail have been attacked in many different exploits over the years, and sendmail makes an excellent example of why certain software packages are targeted. In part, it is the fact that it is a large and complex system that lends itself to different types of attacks, but it is also that sendmail runs everywhere. Many, many systems on *every* major network run sendmail, and the SMTP port that sendmail monitors is often allowed to pass through firewalls. Intruders look for easy targets, and with so many copies of sendmail running on desktop Unix and Linux clients, they are bound to find one that is misconfigured or out–of–date.

However, there is no need to run so many copies of sendmail. A Linux client doesn't need to run a sendmail daemon. It can be configured to relay mail to a departmental server that is running sendmail for outbound mail and to collect its mail from a mailbox server for inbound mail. Simply reducing the number of copies of a software package that you run on your network reduces the chances for a configuration error that is exploited by an intruder.

I'm not trying to pick on sendmail. I could repeat this same argument about IMAP and dozens of other software packages. In fact, most network daemons fall into this category—clients don't need them.

There are two simple ways to block access to unneeded daemons:

Disable daemons from the inetd or xinetd configurations Most network services are started by inetd or xinetd, which only start services listed in their configuration

files. Disabling a service in the configuration file prevents outsiders from using the specified network service, but it does not block the desktop client from using the service on outbound connections. Thus, if ftp is disabled in inetd.conf (or xinetd.conf), the user can still ftp to remote sites, but no one from a remote site can use ftp to log in to the user's desktop. To disable a service in the inetd.conf file, place a hash mark at the beginning of the entry for that service. To prevent xinetd from starting a service, set the parameter disable = yes in that service's xinetd configuration. (On Red Hat systems, the chkconfig command can be used to set the disable parameter for a service; that is, chkconfig imap off sets disable = yes in the /etc/inetd.d/imap configuration file.)

Remove scripts that launch unneeded daemons from the startup Some network service daemons, such as sendmail and named, are started at boot time. Use tksysv or chkconfig to remove unneeded daemons from the startup. For example, a Red Hat desktop client does not need to run the httpd startup script if it is not a web server. The client can still use a web browser, even if the httpd script is not run at startup.

Network daemons are not the only unneeded software, and clients are not the only targets. Unneeded software on a server can also open a hole for an intruder. If you have a dedicated DNS server, it needs to run named, but it doesn't require the sendmail daemon. Likewise, if you're running a dedicated mailbox server, it doesn't need to have a C compiler installed. (You can do your compiles on your desktop system and move the finished product to the server.)

Tip Everything you have on your server is a potential tool for an intruder. Think hard about what you really need for the server to do its job.

There are two ways to limit the software installed on a server. First, when you do the initial Linux installation, don't install what you don't need. During the initial installation, you select the software packages that are installed and the daemons that are loaded. Choose carefully, based on your plan for the system you're installing.

The other way to limit the software on a system is to remove it after it is installed. For example, to remove IMAP from a system with rpm, you could enter **rpm –e imap–2000c–15**.

In addition to installing the latest software and removing unneeded software, you should limit access to the software and services running on your system to just those systems that you actually want to serve. Linux makes this simple by providing access control mechan–isms. Systems that use inetd can use tcpd wrapper software to control access. Systems that use xinetd can use the access–control features included in xinetd. And all systems with the Linux 2.4 kernel can use iptables to limit access.

Controlling Access with *tcpd*

The tcpd wrapper software is executed by inetd. It is an integral part of most Linux distributions that use inetd. Using tcpd on a Linux system is easier than it is on many other systems because the entries in the inetd.conf file already point to the tcpd program.

Note The format of the inetd.conf file is explained in Chapter 3, "Login Services".

The following entries are from the inetd.conf file on a Linux system:

```
ftp    stream tcp  nowait root /usr/sbin/tcpd in.ftpd -l -a
telnet stream tcp  nowait root /usr/sbin/tcpd in.telnetd
shell  stream tcp  nowait root /usr/sbin/tcpd in.rshd
login  stream tcp  nowait root /usr/sbin/tcpd in.rlogind
talk   dgram  udp  wait   root /usr/sbin/tcpd in.talkd
ntalk  dgram  udp  wait   root /usr/sbin/tcpd in.ntalkd
imap   stream tcp  nowait root /usr/sbin/tcpd imapd
finger stream tcp  nowait root /usr/sbin/tcpd in.fingerd
```

As this sample shows, the path to tcpd is used in place of the path of each network service daemon. Therefore, when inetd receives a request for a service, it starts tcpd. tcpd then logs the service request, checks the access control information, and (if permitted) starts the real daemon to handle the request.

The tcpd program performs two basic functions: It logs requests for Internet services, and it provides an access control mechanism for those services. Logging requests for specific network services is a useful monitoring function, especially if you are looking for possible intruders.

Tracking Remote Access

tcpd uses the authpriv facility of syslogd to log its messages. Look in the /etc/syslog.conf file to find out where your system logs authpriv messages. For example, the messages shown in Listing 12.1 are logged to /var/log/secure.

Listing 12.1: The *tcpd* Security Log

```
# cat /var/log/secure
Jun  9 08:09 owl login: ROOT LOGIN ON tty1
Jun  9 08:48 owl login: LOGIN ON tty1 BY craig
Jun 11 00:48 owl in.telnetd[950]: connect from 172.19.50.52
Jun 11 00:48 owl login: LOGIN ON 1 BY craig FROM beaver.example.org
Jun 12 01:11 owl in.telnetd[3467]: connect from 127.0.0.1
Jun 12 01:11 owl login: LOGIN ON 2 BY craig FROM localhost
Jun 12 01:19 owl imapd[3489]: connect from 127.0.0.1
Jun 14 10:23 owl in.telnetd[2090]: connect from 172.19.24.1
Jun 14 10:23 owl login: LOGIN ON 1 BY craig FROM cat.example.org
Jun 15 14:30 owl in.ftpd[10201]: connect from sr1.sybex.com
Jun 16 05:27 owl in.rshd[6434]: connect from 172.19.60.22
Jun 17 20:20 owl login: ROOT LOGIN ON tty1
Jun 17 14:54 owl in.telnetd[1388]: connect from 172.19.50.52
Jun 17 14:54 owl login: LOGIN ON 2 BY craig FROM beaver.example.org
Jun 18 14:28 owl in.ftpd[10190]: refused connect from 172.25.98.2
```

This sample /var/log/secure file shows that not everything in this log comes from tcpd. It also contains messages for login. Combining the two messages can provide some useful insight.

The logins on June 9 are from the system console. The first message from tcpd is the telnet connection on June 11. The message tells you that someone used telnet to connect to owl from IP address 172.19.50.52. The login message that follows tells you that the person logged in as craig, and that the hostname associated with the remote IP address is beaver.example.org. If this is what you expect, there is nothing to be concerned about.

In this particular file, the message that draws our attention occurred on June 16. tcpd reports that someone accessed the system through remote shell (rshd) from IP address 172.19.60.22 on that date. The remote shell can be used to remotely execute commands on your system, so it can be a

powerful tool for intruders. Most systems do not even allow remote shell. If you don't believe that remote shell is configured on your system, if you don't recognize the IP address, or if you don't understand why someone at that address would be running a remote shell to your system, you should be concerned. Watch the log to see if a pattern develops.

Of less concern is the message from June 18 that shows a failed attempt to connect to ftp. This bears watching if it occurs frequently, but it is not yet a problem because the connection was refused based on tcpd wrapper's access–control configuration.

If logging were all it did, tcpd would be a useful package. But the real power of this tool is its ability to control access to network services.

tcpd Access Control Files

Two files define access controls for tcpd:

- The hosts.allow file lists the hosts that are allowed to access the system's services.
- The hosts.deny file lists the hosts that are denied service.

If these files are not found, tcpd allows every host to have access, and simply logs the access request.

When the files are present, tcpd reads the hosts.allow file first and then reads the hosts.deny file. It stops as soon as it finds a match for the host and the service in question. Therefore, access granted by hosts.allow cannot be overridden by hosts.deny. For this reason, it is common to start by first inserting an entry in hosts.deny that denies all access to all systems, and then to continue by placing entries in the hosts.allow file that permit access to only those systems that really should receive services. The format of entries in both files is the same:

services **:** *clients* [**:** *shell–command*]

services is a comma–separated list of network services or the keyword ALL. ALL is used to indicate all network services. Otherwise, each individual service is identified by its process name, which is the name that immediately follows the path to tcpd in the inetd.conf file. For example, the process name in the following inetd.conf entry is imapd:

```
imap stream tcp nowait root /usr/sbin/tcpd imapd
```

clients is a comma–separated list of hostnames, domain names, Internet addresses, network numbers, and the keyword LOCAL. Alternatively, it can be the keyword ALL. ALL matches all hostnames and addresses; LOCAL matches all hostnames that do not include a domain name part. A hostname matches an individual host. An IP address can be defined by itself to match a specific host or with an address mask to match a range of addresses. A domain name starts with a dot (.) and matches every host within that domain. A network number ends with a dot and matches every IP address within the network address space.

shell–command is an optional shell command that tcpd executes when a match occurs. If a match occurs, tcpd logs the access, grants or denies access to the service, and then passes the shell command to the shell for execution.

A few examples can illustrate the variety of valid ways in which *services* and *clients* can be described in a tcpd access–control entry. First, here's something simple from an imaginary

hosts.allow file:

```
ALL : LOCAL, .foobirds.org
in.ftpd,in.telnetd : sr1.sybex.com
```

The keyword ALL in the *services* field indicates that the first rule applies to all network services. In the *clients* field, the keyword LOCAL indicates that all hostnames without a domain part are acceptable, and.foobirds.org matches all hostnames in that domain. By itself, LOCAL would match wren, but not wren.foobirds.org. Combining these two tests in a single rule allows every system in the local domain to use all of the network services. The second rule grants ftp and telnet access to users on the remote system sr1
.sybex.com.

The syntax of a standard tcpd access−control file can be a little more complicated than the preceding example. A hosts.allow file might contain the following:

```
imapd, ipopd3 : 172.5.4.
ALL EXCEPT imapd, ipopd3 : ALL
```

The first entry says that every host whose IP address begins with 172.5.4 is granted access to the IMAP and POP services. The second line says that all services except IMAP and POP are granted to all hosts. These entries would limit mailbox service to a single subnet while providing all other services to anyone that requested them. The EXCEPT keyword is used to except items from an all−encompassing list. It can also be used on the *clients* side of an access rule. For example:

```
ALL: .foobirds.org EXCEPT crow.foobirds.org
```

If this appeared in a hosts.allow file, it would permit every system in the foobirds.org domain to have access to all services except for the host crow.foobirds.org. The assumption is that crow.foobirds.org is not trusted for some reason—perhaps users outside of the domain are allowed to log in to crow.

The final syntax variation uses the at−sign (@) to narrow the definition of *services or clients.* Here are two examples:

```
in.telnetd@172.16.7.2 : 172.16.7.0/255.255.255.0
in.rshd : KNOWN@robin.foobirds.org
```

When the @ appears in the services side of a rule, it indicates that the server has more than one IP address, and that the rule being defined applies only to one of those addresses. Examples of systems with more than one address are multihomed hosts and routers. If your server is also the router that connects your local network to outside networks, you may want to provide services on the interface connected to the local network while not providing those services on the interface connected to the outside world. The @ syntax lets you do that. If the first line in this example appeared in a hosts.allow file, it would permit access to the telnet daemon through the network interface that has the address 172.16.7.2 by any client with an address that begins with 172.16.7.

The purpose of the @ when it appears on the *clients* side of the rule is completely different. On the *clients* side, the @ indicates that a username is required from the client as part of the access control test. This means that the client must run identd. You can test for a specific username, but it's more common to use one of three possible keywords:

> **KNOWN** The result of the test is KNOWN when the client returns a username in response to the query.

UNKNOWN The result of the test is UNKNOWN when the client does not run identd, and thus fails to respond to the query.

ALL This setting requires the client to return a username. It is equivalent to using KNOWN, but is less commonly used.

Defining an Optional Shell Command

The shell command allows you to define additional processing that is triggered by a match in the access control list. In all practical examples, this feature is used in the hosts.deny file to gather more information about the intruder or to provide immediate notification to the system administrator about a potential security attack. For example:

```
in.rshd : ALL : (safe_finger -l @%h | /usr/sbin/mail -s %d - %h root) &
```

In this example from a hosts.deny file, all systems are denied access to rshd. After logging the attempted access and blocking it, tcpd sends the safe_finger command to the shell for execution. All versions of finger, including safe_finger, query the remote host to find out who is logged–in to that host. This information can be useful when tracking down an attacker. The result of the safe_finger command is mailed to the root account. The ampersand (&) at the end of the line causes the shell commands to run in the background. This is important; without it, tcpd would sit and wait for these programs to complete before returning to its own work.

Note The safe_finger program is provided with the tcpd wrapper software. It is specially modified to be less vulnerable to attack than the standard finger program.

There are some variables, such as %h and %d, used in the shell command example. These tcpd wrapper variables, listed in Table 12.1, allow you to take values for the incoming connection and use them in the shell process.

Table 12.1: Wrapper Variables

Variable	Value
%a	The client's IP address.
%A	The server's IP address.
%c	All available client information, including the apparent username when available.
%d	The network service daemon process name.
%h	The client's hostname. If the hostname is unavailable, the IP address is used.
%H	The server's hostname.
%n	The client's hostname. If the hostname is unavailable, the keyword UNKNOWN is used. If a DNS lookup of the client's hostname and IP address do not match, the keyword PARANOID is used.
%N	The server's hostname.
%p	The network service daemon process ID (PID).
%s	All available server information, including the username when available.
%u	The client username or the keyword UNKNOWN if the username is unavailable.
%%	The percent character (%).

Using information from Table 12.1, you can figure out exactly what the sample shell command is doing. Assume that the attempted access to in.rshd came from the host bad.worse.org. The

command passed to the shell is the following:

```
safe_finger -l @bad.worse.org |
   Ä/usr/sbin/mail -s in.rshd-bad.worse.org root
```

With all of the possible rules that can be defined with the standard tcpd access control syntax, you should be able to define the access rules you needed. Yet, it is possible to use an extended version of the tcpd access control language.

Optional Access Control Language Extensions

If tcpd is compiled with PROCESS_OPTIONS enabled in the Makefile, the syntax of the access control language is changed and extended. The extended command syntax is not limited to three fields. The extended syntax is

```
services : clients : option : option
```

The *services* field and the *clients* field are defined in exactly the same way as they were in the original Wrapper syntax. The *option* field is new and so is the fact that multiple options are allowed for each rule. There are several possible options:

allow Grants the requested service. This option must appear at the end of a rule.

deny Denies the requested service. This option must appear at the end of a rule.

spawn *shell–command* Executes the shell command as a child process.

twist *shell–command* Executes the shell command instead of the requested service.

keepalive Sends keepalive messages to the client. If the client does not respond, the connection is closed.

linger *seconds* Specifies how long to try to deliver data after the server closes the connection.

rfc931 [*timeout*] Uses the IDENT protocol to look up the client username. *timeout* defines how many seconds the server should wait for the client's response. The default timeout period is specified as a compiler option.

banners *path* Displays the contents of a message file to the client. *path* is the name of a directory that contains the banner files. The file displayed is the file that has the same name as the network daemon process.

nice [*number*] Sets the nice value for the network service process. The nice value is used to calculate a scheduling priority. The default value is 10.

umask *mask* Sets a umask value for files created by the process. The value defined by umask turns off bits in the default file mode to produce the new permission. For example, assuming that the default file mode is 0666 and the umask value is 022, removing the bits defined by 022 from 0666 produces a file permission of 0644.

user *user[.group]* Runs the network service process with the specified user ID and group ID regardless of what is defined in inetd.conf.

setenv *name value* Sets an environment variable for the process runtime environment.

A few examples based on previous examples illustrate the differences in the new syntax. Using the new syntax, a hosts.allow file might contain

```
ALL : LOCAL, .foobirds.org : ALLOW
in.ftpd,in.telnetd : sr1.sybex.com : ALLOW
ALL : ALL : DENY
```

With the new syntax, there is no need to have two files. The options ALLOW and DENY permit everything to be listed in a single file. The function of the first two lines is identical to the first host.allow example described earlier. The third line is the same as having the line ALL : ALL in the hosts.deny file. Everything done with the basic syntax can be done in a single file with the extended syntax.

Using the ALLOW and DENY options, this command

```
ALL: .foobirds.org EXCEPT crow.foobirds.org
```

can be rewritten as

```
ALL: crow.foobirds.org : DENY
ALL: .foobirds.org : ALLOW
```

The shell command example using the original syntax is almost identical in the new syntax:

```
in.rshd : ALL: spawn (safe_finger -l @%h |
 Ä/usr/sbin/mail -s %d - %h root) & : DENY
```

A more interesting variation on the shell command theme comes from using the twist option. Instead of passing a command to the shell for execution, the twist command executes a program for the client—but not the program the client expects. For example:

```
in.ftpd : ALL: twist /bin/echo 421 FTP not allowed from %h : DENY
```

In this case, when the remote system attempts to start the FTP daemon, echo is started instead. The echo program then sends the message to the remote system and terminates the connection.

Other programs, such as portmapper, use the host.allow and host.deny file. Not all of these programs understand the extended syntax. For this reason, most system administrators stick with the basic syntax. See the sidebar "Realistic Wrapper Rules" for an example of a common tcpd wrapper configuration.

Realistic Wrapper Rules

The most important thing you can do to secure your system is to keep the software up-to-date. This is essentially a passive activity—you depend on the software developer for the update. A more active approach to security is access control—it is something you can do even before problems are detected. The two basic rules you use to configure access controls are very simple:

- Don't run any services you don't need to run.
- Don't provide service to anyone to whom you don't need to provide service.

These rules lead to the following simple tcpd configuration that is put on many ser–vers immediately after installing the system software. First, to prevent access from everyone in the outside world, make this entry in the /etc/hosts.deny file:

```
ALL : ALL
```

Next, make an entry in /etc/hosts.allow to permit access to the necessary services by clients in the local domain:

```
ALL : LOCAL, .foobirds.org
```

This example assumes that the local domain is foobirds.org and that this server provides all of its services to the local domain. This is often the case because unneeded software is either not installed or blocked from running. Therefore, all of the software left running on the server is intended to serve clients.

This simple configuration is effective against many attacks. Attack scripts look for simple targets, and they are often discouraged by the first sign of resistance. Of course, a skilled and determined attacker can find his way around any defense. Luckily, most of us don't really attract attacks from highly skilled people.

Controlling Network Access with *xinetd*

xinetd is the alternative to inetd used on some Linux systems—Red Hat Linux, for example. xinetd is configured in the /etc/xinetd.conf file. Chapter 3 describes the basic syntax and structure of the xinetd.conf file. As explained there, Red Hat creates individual xinetd configuration files for each service in the /etc/xinetd.d directory. These individual files are included by reference into the xinetd.conf file. Listing 12.2 shows a sample file from the xinetd.d directory.

Listing 12.2: An *xinetd* Configuration File

```
# default: off
# description: The IMAP service allows remote users to access their
#              mail using an IMAP client such as Mutt, Pine, fetchmail,
#              or Netscape Communicator.
service imap
{
        socket_type             = stream
        wait                    = no
        user                    = root
        server                  = /usr/sbin/imapd
        log_on_success          += DURATION USERID
        log_on_failure          += USERID
        disable                 = no
}
```

The service, socket_type, wait, user, and server values all parallel values found in the inetd.conf file, except that the path provided for the server parameter does not invoke the tcpd wrapper program. The wrapper is not used because xinetd provides capabilities similar to those of wrapper on its own.

xinetd provides its own logging and its own access controls. Despite the fact that xinetd does not invoke tcpd, it does consult the /etc/hosts.allow and /etc/hosts.deny files in addition to the access controls defined in the xinetd.conf file. This means that hosts.allow and hosts.deny files created for other programs, such as portmapper, can provide security guidance to xinetd.

The log_on_success and log_on_failure lines in Listing 12.2 add the user ID of the remote user to the standard log entry when a successful connection is made or a connection attempt fails. The log_on_success line also logs the length of time that the server handling this connection ran. (DURATION applies only to log_on_success.) The += syntax means that the values defined for log_on_success and log_on_failure are added to the other values already being logged. In addition to logging the duration of the connection and the user ID of the remote user, log_on_success and log_on_failure allow you to log the following:

HOST The address of the remote host. Like USERID, this value can be used for both success and failure.

PID The process ID of the server started to handle the connection. PID applies only to log_on_success.

EXIT Logs the exit status of the server when the connection terminates. EXIT applies only to log_on_success.

ATTEMPT Logs unsuccessful connection attempts. ATTEMPT applies only to log_on_ failure.

RECORD Logs the connection information received from the remote server. This parameter applies only to log_on_failure.

The disable = no line in Listing 12.2 means that this service is enabled. To prevent this service from running, set disable to yes. As described earlier, Red Hat allows you to set the disable parameter from the chkconfig command line.

In addition to controlling whether or not a service is available, xinetd provides finer access controls. xinetd provides three different attributes for access control. It can be configured to accept connections from certain hosts, paralleling the host.allow file; to reject connections from certain hosts, paralleling the host.deny file; and to accept connections only at certain times of the day. These attributes are described as follows:

only_from This attribute identifies the hosts that are allowed to connect to the service. Hosts can be defined using

- a numeric address. For example, 172.16.12.5 defines a specific host, and 129.6.0.0 defines all hosts with an address that begins with 129.6. The address 0.0.0.0 matches all addresses.
- an address scope. For example, 172.16.12.{3,6,8,23} defines four different hosts: 172.16.12.3, 172.16.12.6, 172.16.12.8, and 172.16.12.23.
- a network name. The network name must be defined in the /etc/networks file.
- a canonical hostname. The IP address provided by the remote system must reverse–map to this hostname.
- a domain name. The hostname returned by the reverse lookup must be in the specified domain. For example, the value .foobirds.org requires a host in the foobirds.org domain. Note that when a domain name is used, it starts with a

dot.

- an IP address with an associated address mask. For example, 172.16.12.128/25 would match every address from 172.16.12.128 to 172.16.12.255.

no_access This attribute defines the hosts that are denied access to the service. Hosts are defined using exactly the same methods as those described previously for the only_from attribute.

access_times This attribute defines the time of day a service is available, in the form *hour*:*min*–*hour*:*min*. A 24–hour clock is used. Hours are 0 to 23, and minutes are 0 to 59.

If neither only_from nor no_access is specified, access is granted to everyone. If both are specified, the most exact match applies—for example:

```
no_access            = 172.16.12.250
only_from            = 172.16.12.0
```

The only_from command in this example permits every system on network 172.16.12.0 to have access to the service. The no_access command takes away that access for one system. It doesn't matter whether the no_access attribute comes before or after the only_from attribute. It always works the same way because the more exact match takes precedence.

Listing 12.3 shows the /etc/xinetd.d/imap entry from our sample Red Hat system with some access controls added.

Listing 12.3: *xinetd.conf* Access Controls

```
# default: off
# description: The IMAP service allows remote users to access their
#              mail using an IMAP client such as Mutt, Pine, fetchmail,
#              or Netscape Communicator.
service imap
{
        socket_type             = stream
        wait                    = no
        user                    = root
        server                  = /usr/sbin/imapd
        log_on_success          += DURATION USERID
        log_on_failure          += USERID
        disable                 = no
        only_from               = 172.16.12.0
        no_access               = 172.16.12.231
        server                  = /usr/sbin/in.rlogind
}
```

In Listing 12.3, the only_from attribute blocks access from every system except those on network 172.16.12.0. At the same time, it permits access from every system on network 172.16.12.0, which is the local network for this sample system. In Listing 12.3, that is not exactly what we want. There is one system, 172.16.12.231, which is not trusted to have login access. The no_access attribute denies access to anyone on the system 172.16.12.231.

tcpd wrapper can only protect services started by inetd or services, such as portmapper, which read the hosts.allow and hosts.deny file on their own. xinetd can secure only the services it starts. Other

services, such as dhcpd, are started at boot time. To control access to services that are started by boot scripts and that don't read the tcpd configuration file, use the Linux IP firewall.

Controlling Access with *iptables*

Everyone thinks they know what a firewall is until you get down to the details. In a large sense, a *firewall* is a system that protects the local network from the big bad global network. It is the sentinel through which all network traffic must pass before it can enter or exit the local network. In its simplest incarnation, a firewall is a filtering router that screens out unwanted traffic. And at its most complex, it is an entire network with multiple routers and multiple servers.

Linux provides the traffic–filtering tools needed to create a simple firewall. Combining the routing capabilities of Linux with the filtering features of iptables creates a filtering router. Additionally, and more commonly, iptables can be used to filter traffic that arrives at a Linux server's network interface before that traffic is passed up to the network applications running on that server. This gives Linux the capability to build a firewall within the server itself, which provides access control for all possible network services.

Maintaining Firewall Rules with *iptables*

The Linux kernel categorizes firewall traffic into three groups, and applies different filter rules to each category of traffic:

Input firewall Incoming traffic is tested against the input firewall rules before it is accepted.

Output firewall Outbound traffic is tested against the output firewall rules before it is sent.

Forwarding firewall Traffic that is being forwarded through the Linux system is tested against the rules for the forwarding firewall.

The INPUT and OUTPUT rulesets can be used when the system is acting as a host. The FORWARD rules are used when the system acts as a router. In addition to the three standard firewall categories, the Linux kernel provides the network address translation rulesets described in Chapter 7, "Network Gateway Services," and allows user–defined categories.

The Linux kernel maintains a list of rules for each of these categories. These lists of rules, called *chains*, are maintained by the iptables command. Use the following options with the iptables command to create or delete user–defined rules, add rules to a chain of rules, delete rules from a chain, and change the order of the rules in the chain:

–A Appends rules to the end of a chain.

–D Deletes selected rules from a chain.

–E Renames a chain.

–F Removes all of the rules from a chain.

–I Inserts rules into a chain. A rule number is defined to specify where in the chain of rules the new rule is inserted. For example, to insert a rule at the head of the chain, assign it rule number 1.

–L Lists all rules in a chain. If no chain is specified, all rules in all chains are listed.

–N Creates a user–defined chain with the specified name.

–P Sets the default policy for a chain.

–R Replaces a rule in a chain.

–X Deletes the specified user–defined chain.

–Z Resets the packet and byte counters in all chains to zero.

Firewall rules are composed of a filter against which the packets are matched and the action taken when a packet matches the filter. The action can either be a standard policy or a jump to a user–defined rule chain for additional processing. The –j *target* command–line option defines the action to be taken. *target* can be either the name of a user–defined rule or a standard policy. The *target* keywords that identify the standard policies are the following:

accept Lets the packet pass through the firewall.

drop Discards the packet.

queue Passes the packet up to user space for processing.

return In a user–defined rule chain, return means to return to the chain that called this chain. In one of the three kernel chains, it means to exit the chain and use the default policy for the chain.

Use the parameters that come with the iptables command to construct filters that match the protocol used, the source or destination address, or the network interface used for the packet. The iptables parameters are as follows:

–p *protocol* Defines the protocol to which the rule applies. *protocol* can be any numeric value from the /etc/protocols file or one of the following keywords: tcp, udp, or icmp.

–s *address[/mask]* Defines the packet source to which the rule applies. *address* can be a hostname, a network name, or an IP address with an optional address mask.

––sport[*port*[:*port*]] Defines the source port of the packets to which this rule applies. *port* can be a name or number from the /etc/services file. A range of ports can be specified using the format *port*:*port*. If no specific port value is specified, all ports are assumed.

–d *address[/mask]* Defines the packet destination to which the rule applies. The address is defined using the same rules as those used to define the address for the packet source.

−−dport [*port*[:*port*]] Defines the destination port to which the rule applies. This filters all traffic bound for a specific port. The port is defined using the same rules as those used to define these values for the packet source.

−−icmp−type *type* Defines the ICMP type to which the rule applies. *type* can be any valid ICMP message type number or name.

−j *target* Identifies a standard policy to handle the packet or a user−defined chain to which control should be passed.

−i *name* Defines the name of the input network interface to which the rule applies. Only packets received on this interface are affected by this rule. A partial interface name can be used by ending it with a +; for example, eth+ would match all Ethernet interfaces that begin with eth.

−o *name* Defines the name of the output network interface to which the rule applies. Only packets sent on this interface are affected by this rule. The same interface name rules used with the −i option are used with this option.

−f Indicates that the rule refers only to second and subsequent fragments of fragmented packets.

Sample *iptables* Commands

Putting this all together creates a firewall that can protect your network. Assume that we have a Linux router attached to a perimeter network with the address 172.16.12.254 on interface eth0 and to an external network with the address 192.168.6.5 on interface eth1. Further assume that the perimeter network contains only a sendmail server and an Apache server. Listing 12.4 contains some iptables commands we might use on the Linux system to protect the perimeter network.

Listing 12.4: Sample *iptables* Commands

```
iptables -F INPUT
iptables -F FORWARD
iptables -A INPUT -i eth1 -j DROP
iptables -A FORWARD -i eth1 -s 172.16.0.0/16 -j DROP
iptables -A FORWARD -o eth1 -d 172.16.0.0/16 -j DROP
iptables -A FORWARD -d 172.16.12.1 25 -j ACCEPT
iptables -A FORWARD -d 172.16.12.6 80 -j ACCEPT
iptables -A FORWARD -j DROP
```

These first two commands use the −F option to clear the chains we plan to work with. The third line drops any packets from the external network that are bound for a process running locally on the Linux router. We do not allow any access to router processes from the external world.

The next two commands drop packets that are being routed to the external world using an internal address. If packets are received on the external interface with a source address from the internal network, they are dropped. Likewise, if packets are being sent out the external interface with a destination address from the internal network, they are dropped. These rules say that if packets on the external network interface (which is eth1 in the example) misuse addresses from the internal network (which is 172.16 in the example), then somebody is trying to spoof us, and the packets should be discarded.

The next two rules are basically identical. They accept packets if the destination and port are the correct destination and port for a specific server. For example, port 25 is SMTP, and 172.16.12.1 is the mail server; and port 80 is the HTTP port, and 172.16.12.6 is the web server. We accept these inbound connections because they are destined for the correct systems. The last rule rejects all other traffic.

These examples illustrate the power of Linux's built–in filtering features, and provide enough information to get you started. Clearly, much more can and should be done to build a real firewall. If you want to know more about iptables, see *Linux Security* by Ramón Hontañón (Sybex, 2001) and *Building Internet Firewalls* by Elizabeth Zwicky, Simon Cooper, and D. Brent Chapman (O'Reilly, 2000) for many more detailed iptables examples. If you have an older Linux system that uses either ipchains or ipfwadm, see *Linux Firewalls* by Robert Ziegler (New Riders, 2001) for information about these older commands.

Improving Authentication

Traditional Unix passwords are no more than eight characters long and are transmitted across the network as clear text. Additionally, these passwords are stored in the /etc/passwd file, which is world–readable. All of these things are security problems.

Limiting passwords to eight characters limits a user's choices, and reduces the difficulty of a brute–force decryption attack. MD5 passwords can be up to 256 characters long. On a Red Hat system, select MD5 Passwords during the installation, as described in Appendix A, "Basic Installation." Alternatively, you can run authconfig ––enablemd5 after the system is running to enable the use of long passwords.

Regardless of how long a password is, the user can pick a bad one. A bad password is one that is easy to guess. See the following "Password Dos and Don'ts" sidebar for some advice you can give your users to help them pick good passwords.

Password Dos and Don'ts

- use a mixture of numbers, special characters, and mixed–case letters.
- Do use at least eight characters.
- Do use a seemingly random selection of letters and numbers that is easy to remember, such as the first letter of each word from a line in a book, song, or poem.
- Don't use the name of a person or a thing.
- Don't use any English or foreign language word or abbreviation.
- Don't use any information associated with the account, such as the login name, the user's initials, phone number, social security number, job title, or room number.
- Don't use keyboard sequences; for example, qwerty.
- Don't use any of the bad passwords described above spelled backward, in caps, or otherwise disguised.
- Don't use an all–numeric password.
- Don't use a sample password, no matter how good, that you've gotten from a book that discusses computer security.

Linux prevents users from picking the worst kinds of passwords by applying many of the rules listed in the sidebar to reject bad passwords. Passwords are chosen with the passwd command. Linux tests the password entered by the user at the passwd prompt in several different ways. Listing 12.5

is an example of Red Hat using the pam_cracklib module to block the selection of some bad passwords.

Listing 12.5: Linux Rejects Weak Passwords

```
$ passwd
Changing password for craig
(current) UNIX password:
New UNIX password:
BAD PASSWORD: it is derived from your password entry
New UNIX password:
BAD PASSWORD: it is too simplistic/systematic
New UNIX password:
BAD PASSWORD: it is too short
passwd: Authentication token manipulation error
```

Although Linux does its best to make sure you use a good password, no matter how good a password is, it is useless if someone steals it. Because the passwords are transmitted over the network as clear text, they are very easy to steal.

Two packages that can prevent thieves from stealing passwords off of the wire are described later in this section. However, passwords do not have to be stolen off of the wire. If passwords are stored in the /etc/passwd file, the entire file can be read by anyone on the system and subjected to a "dictionary attack." In a dictionary attack, a large selection of possible passwords are encrypted using the same method of encryption that is used for passwords, and the result of the encryption is compared to the passwords stored in the /etc/passwd file. When the encrypted values match, you know the original password because you know the string you used to create the encrypted value.

Even when good encryption is used for passwords stored in the passwd file, if passwords are poorly chosen, they are susceptible to a dictionary attack. The first line of defense against this problem is to store the encrypted passwords in a file that is not world–readable.

Shadow Passwords

The shadow password file, /etc/shadow, can be read only by root. It grants no world or group file permissions. It is designed to prevent ordinary users from reading the encrypted passwords and subjecting them to a dictionary attack. In Appendix A, this feature is enabled by selecting Enable Shadow Passwords during the Red Hat installation—in the same window in which MD5 passwords are enabled. On a running Red Hat system, authconfig ––enableshadow can be used to enable the use of the shadow password file.

In addition to improved password security, the shadow password file provides the system administrator with some password–management features. The shadow password file contains encrypted passwords and the information needed to manage them. The format of a shadow password file entry is the following:

```
username:password:changed:min:max:warn:inactive:close:reserved
```

In this entry,

- *username* is the login username.
- *password* is the encrypted password.

- *changed* is the date that the password was last changed, written as the number of days from January 1, 1970, to the date of the change.
- *min* is the minimum number of days the user must keep a new password before it can be changed.
- *max* is the maximum number of days the user is allowed to keep a password before it must be changed.
- *warn* is the number of days that the user is warned before the password expires.
- *inactive* is the number of days after the password expires before the account is locked. After the account is locked, the user is not able to log in and change his password.
- *close* is the date on which the account will be closed, written as the number of days from January 1, 1970, to the date that the account will be closed.
- *reserved* is a field reserved for the system's use.

An excerpt from the shadow password file on a Red Hat system is shown in Listing 12.6.

Listing 12.6: Excerpts from the Shadow Password File

```
root:$1$1yBKhGuF$xocwED2RSGT03jEtq4yJ0/:11530:0:99999:7:::
bin:*:11530:0:99999:7:::
daemon:*:11530:0:99999:7:::
adm:*:11530:0:99999:7:::
uucp:*:10750:0:99999:7:::
nobody:*:11530:0:99999:7:::
nscd:!!:11530:0:99999:7:::
mailnull:!!:11530:0:99999:7:::
ident:!!:11530:0:99999:7:::
rpc:!!:11530:0:99999:7:::
rpcuser:!!:11530:0:99999:7:::
xfs:!!:11530:0:99999:7:::
gdm:!!:11530:0:99999:7:::
craig:$1$W/j5NklD$J.wD9I/toKet.:11530:0:99999:7:::
kathy:$1$iugiomsnsi/ufdjhbhjbih:11720:0:99999:7:::
sara:$1$piuhihblhj./ddkibhtyjjt:11751:0:99999:7:::
david:$1$kjiojhjhjkhplttw3vjhvu:11751:0:99999:7:::
rebecca:$1$ihiohuhxvf56/uhhfhjH:11751:0:99999:7:::
```

The encrypted password appears only in this file. Every password field in the /etc/passwd file contains an x, which tells the system to look in the shadow file for the real password. Every password field in the /etc/shadow file contains either an encrypted password, !!, or *. If the password field contains !!, it means that the account has a valid login shell, but the account is locked so that no one can log in through the account. If the password field contains *, it indicates that this is a system account, such as daemon or uucp, which does not have a login shell and therefore is not a login account.

Password Aging

In addition to protecting the password, the shadow file supports password aging, which defines a lifetime for each password and notifies the user to change the password when it reaches the end of its life. If it is not changed, the user is blocked from using her account. The changed, max, and warn fields tell the system when the password was changed, how long it should be kept, and when to warn the user to change it.

When the password is changed, it must be used for the number of days defined by the min field before it can be changed again, which prevents the user from changing his favorite password to a

temporary password and then immediately back to the favorite. This reduces one of the most common tricks used to avoid really changing passwords.

The inactive and close fields help eliminate unused accounts. The inactive field gives the user some number of days to log in and set a new password after the password expires. If the user does not log in before the specified number of days has elapsed, it indicates that the account is unused, and the account is locked to prevent the user logging in.

The close field lets you create a user account that has a specified "life." When the date stored in the close field is reached, the user account is disabled, even if it is still active. The expiration date is stored as the number of days since January 1, 1970.

On most Linux systems, some of the values in /etc/shadow file can be modified by using the useradd or usermod commands. Listing 12.7 shows how the usermod command can be used to set the inactive field and the close field in the shadow password file.

Listing 12.7: Modifying */etc/shadow* with *usermod*

```
[root]# grep alana /etc/shadow
alana:RoseySmile:11743:0:99999:7:::
[root]# usermod -f 30 alana
[root]# grep alana /etc/shadow
alana:RoseySmile:11743:0:99999:7:30::
[root]# usermod -e 2003-01-01 alana
[root]# grep alana /etc/shadow
alana:RoseySmile:11743:0:99999:7:30:12053:
```

The first grep command shows that an entry exists for the user alana in the /etc/shadow file. The *min*, *max* and *warn* values have already been set to 0, 99999, and 7, respectively. A *min* value of 0 means that the user does not need to wait before changing the password. A *max* value of 99999 means that the user is never forced to change the password, and a *warn* value of 7 means the user is warned one week before the password expires. (With a *max* of 99999, the *warn* field will not really be used.) As explained in Chapter 3, these values are set by the useradd command using the values defined for the PASS_MIN_DAYS, PASS_MAX_DAYS, and PASS_WARN_AGE variables in the /etc/login.defs file. To set different defaults, edit /etc/login.def, and set different values for the variables. To change these fields for an individual user in the /etc/shadow file, edit the shadow file and change the values there.

The first usermod command shows how the *inactive* value is set using the –f command line argument. In Listing 12.7, the inactive field is set to 30 days, as the second grep clearly shows.

The last usermod command sets the value for *close*. Generally, this is used instead of the other values for short–term user accounts. For most accounts, this field is not used. In Listing 12.7, close is set so that the alana account will be closed on January 1, 2003.

At many sites, the default values, which do not implement password aging, are used. Password aging has only a marginal value for increasing system security. It is primarily used to remind users about security and to make sure that unused accounts do not go unnoticed. The most important feature of the shadow password file is that it protects the passwords that are stored on your system from a dictionary attack. However, it does not prevent the passwords from being stolen off the wire when they are transmitted. Other tools exist to protect passwords as they pass across the network.

One–Time Passwords

Choosing good passwords and protecting the password file are useless if a thief steals the password from the network. Clear–text, reusable passwords that travel over a network simply aren't secure. All security experts know this, so several alternatives to reusable passwords have been created. One of these is a *one–time password*, which is just what it sounds like—you use the password once and throw it away. These passwords are desirable because they cannot be reused. Anyone who steals a one–time password is stealing useless garbage.

One–time Passwords In Everything (OPIE) is free, one–time passwords software for Linux. OPIE is available from http://www.inner.net/ in the /pub/opie directory. OPIE is source code delivered in a compressed tar file opie–2.4.tar.gz.

Installing OPIE replaces login, su, and ftpd with its own versions of these programs that accept both traditional passwords and OPIE one–time "password phrases"—a string of six short "words," which may or may not be real words.

The OPIE Transition Mechanism

OPIE can be configured to accept either traditional reusable passwords or OPIE password phrases. Users like this feature because they can use convenient reusable passwords for local console logins for which there is no danger of having the password stolen, and can use one–time passwords for remote logins. The problem with this feature is that it opens up a very big security hole by making it possible for people to forget what they are doing and use a reusable password in the wrong situation.

Sometimes, however, you need to use this feature to overcome the resistance to one–time passwords. To enable this feature, run configure with ––enable–access–file when you build the OPIE software, which permits you to use the /etc/opieaccess file. In this file, list the hosts from which reusable passwords are allowed. For example:

```
permit  127.0.0.1      255.255.255.255

deny    172.16.5.25    255.255.255.255

permit  172.16.5.0     255.255.255.0
```

The first field can either permit access with reusable passwords or explicitly deny it. By default, every system not mentioned in the /etc/opieaccess file is denied reusable password access. The second field is the address. The third field is the address mask, which allows you to specify entire networks with a single line.

Selecting Your Secret Password

The list of one–time password phrases is generated by a program named opiekey. To uniquely identify yourself to that program, you need a secret password. Use opiepassword to select that secret password.

For example, assume that I'm new to OPIE, and I want to generate a list of password phrases before going on a trip. First, I log in to the OPIE server's console with my traditional reusable password and run opiepasswd to select a secret OPIE password, which must be at least 10

characters long. opiepasswd accepts the secret password, and displays the first password phrase, which is DUG AHOY EMIL SAM JOT BERN:

```
$ opiepasswd -c
Updating craig:
Reminder - Only use this method from the console;
NEVER from remote. If you are using telnet, xterm, or a dial-in,
type ^C now or exit with no password. Then run opiepasswd without
the -c parameter.
Using MD5 to compute responses.
Enter old secret pass phrase: OJ1CCFftNt
Enter new secret pass phrase: N'pim.c,.na.o
Again new secret pass phrase: N'pim.c,.na.o
ID CRAIG OPIE key is 499 P18318
DUG AHOY EMIL SAM JOT BERN
```

Note Running opiepasswd from the console is the most secure method. If it is not run from the console, you must have a copy of the opiekey software with you to generate the correct responses needed to enter your old and new secret passwords because clear–text passwords are accepted only from the console.

Creating Additional Password Phrases

One password phrase, of course, is not enough. To generate additional password phrases, run opiekey. The second–to–last line output by the opiepasswd command contains important information. It displays the initial sequence number (499) and the seed (p18318). Along with the secret password, these values are required by opiekey to generate the OPIE password phrases.

opiekey takes the login sequence number, the user's seed, and the user's secret password as input; then outputs the correct password phrases. Use the –n argument to request several passwords. Print them out or write them down, and you're ready to go on the road. The example in Listing 12.8 requests five password phrases from opiekey.

Listing 12.8: Generating OPIE Password Phrases

```
$ opiekey -n 5 499 p18318
Using MD5 algorithm to compute response.
Reminder: Don't use opiekey from telnet or dial-in sessions.
Enter secret pass phrase: N'pim.c,.na.o
495: NERO BORN ABET HELL YANG WISE
496: VERB JUKE BRAN LAWN NAIR WOOL
497: POE MOOR HAVE UN DRAB MONT
498: SACK WAND WAKE AURA SNUG HOOD
499: DUG AHOY EMIL SAM JOT BERN
```

Note Login sequence numbers count down from 499 and cannot be reused. When the sequence number gets down to 10, rerun opiepasswd, and select a new secret password in order to reset the sequence number to 499.

The opiekey command line requests five password phrases (–n 5), starting from sequence number 499. The seed (p18318) is provided on the command line. opiekey prompts for the secret password you defined with the opiepasswd command. The sequence number, the seed, and the secret password are then used to generate the password phrases, and opiekey prints out the number you requested in sequence number order.

To log in, you must use the password phrase that goes with the sequence number displayed by login. For example:

```
login: craig
otp-md5 496 p18318
Response or Password: VERB JUKE BRAN LAWN NAIR WOOL
```

A system running OPIE displays a line indicating that the one–time passwords are being generated with the MD5 algorithm (otp–md5), this is login sequence number 496 and the seed used for the one–time passwords is p18318. The correct response is the six short "words" for login number 496 from the list of password phrases. That response cannot be used again to log in to the system.

I may be the only person who will ever tell you, "I like one–time passwords." But I do like them. I particularly like their portability. I don't need any special software on the client, just a list of passwords in my wallet that I can use anywhere. If you have opiekey software on the client system, you can produce one–time passwords, one at a time. But why would you? If you control the software on both the client and on the server, use secure shell. It is a better authentication tool than OPIE. Use OPIE for those times when you need complete freedom to log in from any system. Use secure shell when you want a really good remote login tool.

Secure Shell

The secure shell (SSH) program provides remote access with strong authentication and public key session encryption. Secure shell is both secure and easy to use. SSH replaces telnet, ftp, rlogin, and rsh with secure alternatives, and it is the default remote login tool on our sample Red Hat system. On many Linux systems, SSH is installed as part of the initial system installation. Figure 12.5 shows a gnorpm query of the SSH package on our Red Hat system.

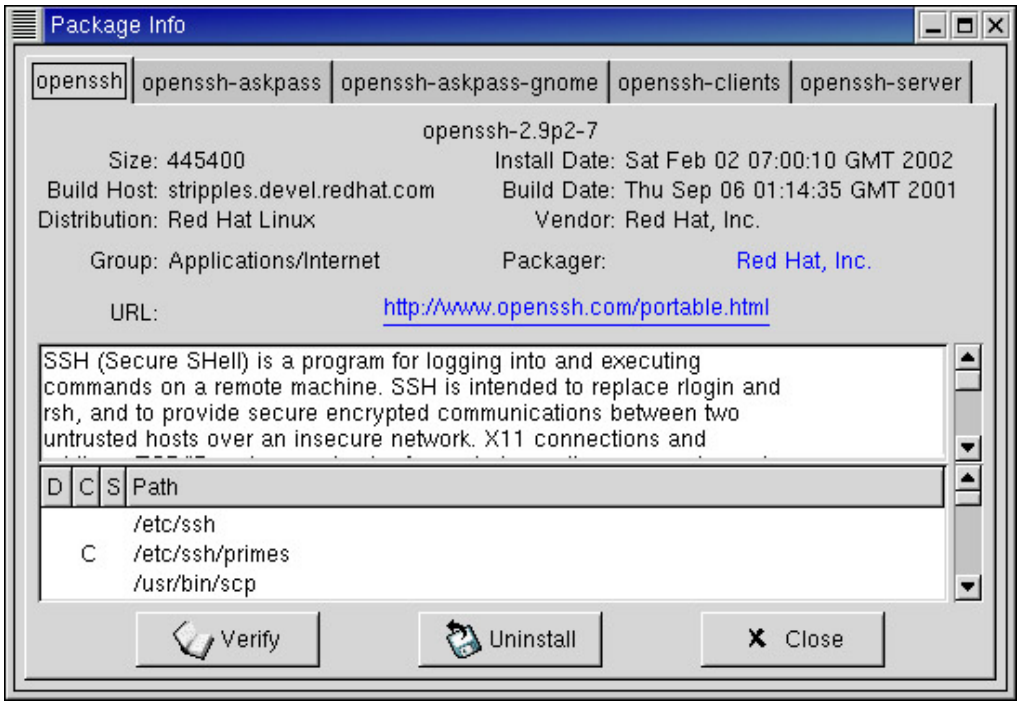


Figure 12.5: The OpenSSH RPM

As Figure 12.5 shows, our sample system uses OpenSSH. SSH was originally developed at Helsinki University of Technology (HUT). Both commercial SSH packages and the open–source OpenSSH software evolved from the original HUT software. Most Linux systems use the OpenSSH package, and that is the software described in this section.

The tabs in Figure 12.5 show that there are five different RPMs used for OpenSSH on our Red Hat system. Two of these, the RPMs that include the string askpass in their names, contain the files

necessary to format the SSH passphrase prompt for the X Windows System. One handles plain X, and the other formats the prompt for the GNOME desktop environment. The real meat of the OpenSSH system comes in the three other packages:

openssh Contains the key generation utility ssh−keygen and the remote file copy program scp.

openssh−server Contains the server daemon sshd and the secure ftp server.

openssh−clients Contains the OpenSSH client tools for client key maintenance, as well as the ssh command for secure login and the sftp command for secure FTP.

When a secure shell client and server connect, they exchange keys. The keys are compared to the known keys. If the key is not found, the user is asked to verify that the new key should be accepted. If the key is accepted by the user, the host key is added to the .ssh/known_hosts file in the user's home directory and then is used to encrypt a randomly generated session key. The session key is then used by both systems to encrypt the remainder of the session. If no special authentication has been configured, the user is prompted for a password; there is no need to worry about password thieves because the password is encrypted before it is sent. Listing 12.9 illustrates how the first login to duck looks from our sample Red Hat system with the default configuration.

Listing 12.9: A Sample *ssh* Login

```
$ ssh duck
The authenticity of host 'duck (172.16.8.8)' can't be established.
RSA key fingerprint is 41:86:62:fb:6e:9f:13:9f:0d:6b:95:d7:09:00:10:a7.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'duck,172.16.8.8' (RSA) to the list of known hosts.
craig@duck's password: Wats?Watt?
[duck]$ logout
Connection to duck closed.
```

The client user is not limited to simple password authentication. By default, the server configuration is set to accept password authentication and public key authentication. If users wish to use public key authentication, they must create their own private and public keys.

Creating User Keys

The ssh−keygen program generates the public and private encryption keys used for public key authentication. Simply invoke the ssh−keygen command and enter a *passphrase*, which is your secret password, when prompted. Listing 12.10 is an example.

Listing 12.10: An Example of the *ssh−keygen* Command

```
$ ssh-keygen
Generating public/private rsa1 key pair.
Enter file in which to save the key (/home/craig/.ssh/identity):
Enter passphrase (empty for no passphrase): Who are the trusted?
Enter same passphrase again: Who are the trusted?
Your identification has been saved in /home/craig/.ssh/identity.
Your public key has been saved in /home/craig/.ssh/identity.pub.
The key fingerprint is:
f1:f4:21:07:ed:8b:fe:3c:66:6a:4d:ff:36:2a:f2:c9
 Äcraig@wren.foobirds.org
```

A passphrase can literally be a phrase because blanks are allowed. The example uses a line from a song. Though a random collection of words may be more secure, the passphrase must be easy to remember. If you forget it, no one will be able to recover it for you.

Two keys are created: ~/.ssh/identity is the private key that must be protected, and ~/.ssh/identity.pub is the public key that is distributed to remote sites for encrypting the session. After the keys are created on the client system, add the complete contents of the identity.pub file to the ~/.ssh/authorized_keys file in the user's home directory on the server. Now, when the user logs in to the server from that client, the user is prompted for the passphrase.

The default configuration provides simple and secure communication, and gives complete protection from password thieves. There are, however, configuration options for both the server and the client that allow you to customize secure shell for your environment.

Configuring *sshd*

Very little configuration is required to get secure shell running, but a great deal of configuration is possible. Many of the software packages discussed in this book fit this pattern: There are a great many configuration options, but the default values for those options work in almost every case and rarely need to be changed. sshd is no exception. Without configuration, it will work just fine, but there are configuration options that you can use to modify it for your particular site.

sshd is configured by the /etc/ssh/sshd_config file. The configuration values in this file are keyword value pairs. There are more than 50 possible configuration values:

AFSTokenPassing Specifies whether or not AFS tokens are accepted by the server.

AllowGroups Lists the groups from which logins are allowed. A user must belong to one of the listed groups in order to log in to the system. By default, logins are not limited to specific groups.

AllowTcpForwarding Specifies whether or not forwarding is permitted. By default, it is permitted.

AllowUsers Lists the users who are allowed to login. By default, logins are not limited to specific users.

Banner Identifies a file that contains a banner page that is displayed before login.

ChallengeResponseAuthentication Specifies whether or not challenge/response authentication should be used. The default is yes, but sshd uses challenge/response authentication only for S/Key.

Ciphers Identifies the encryption techniques that will be supported by the server. The default is aes128–cbc, 3des–cbc, blowfish–cbc, cast128–cbc, and arcfour.

CheckMail Specifies whether or not sshd should check to see whether the user has new mail when the user logs in. By default, it doesn't check.

ClientAliveInterval Defines the time interval at which messages will be sent to the

client when the link is idle. These messages are meant to elicit a response from the client. The default is 0, meaning that the server will not send the messages.

ClientAliveCountMax Defines the maximum number of messages that will be sent to an unresponsive client before the connection is terminated by the server. The default is 3.

DenyGroups Lists the groups that are not allowed to log in to the server. Users belonging to any of these groups will not be allowed to login. By default, all groups are allowed to log in.

DenyUsers Lists the users that are not allowed to log in to the server. By default, all valid users are allowed to log in.

GatewayPorts Specifies whether or not a remote host may connect to a port forwarded for the client. The default is no.

HostbasedAuthentication Specifies whether or not .rhosts and /etc/hosts.equiv will be used to authentication clients. The default is no. .rhosts and /etc/hosts.equiv are the trusted host files traditionally used by rlogin. In this case, however, they are not used alone. The files are combined with public key authentication of the client.

HostKey Identifies the file that contains the host's private keys. The default is /etc/ssh/ssh_host_key.

IgnoreRhosts Means that the .rhosts and .shosts files will not be used, even if RhostsAuthentication, RhostsRSAAuthentication, or HostbasedAuthentication are specified. The /etc/hosts.equiv and /etc/ssh/shosts.equiv files are still used. The default is to ignore .rhosts and .shosts.

IgnoreUserKnownHosts Causes the server to ignore the user's ~/.ssh/known_hosts during RhostsRSAAuthentication and HostbasedAuthentication authentication. The default is no— use the known_hosts file.

KeepAlive Specifies whether or not the system should send keepalive messages to detect whether the link is operational. The default is yes.

KerberosAuthentication Specifies whether or not Kerberos authentication is allowed. The default is yes.

KerberosOrLocalPasswd Tells sshd to use local password authentication if the Kerberos server fails to authenticate the password. The default is yes.

KerberosTgtPassing Specifies whether or not the server will accept Kerberos TGT tokens. The default is no because this works for sshd only when the token is actually an AFS token.

KerberosTicketCleanup Tells the system to destroy the user's Kerberos ticket when the user logs out. The default is yes.

KeyRegenerationInterval Tells sshd to regenerate the ephemeral server key used in SSH version 1 after the specified number of seconds. The default is 3600.

ListenAddress Identifies the address and port on which sshd should listen for connections. The default is to listen on all addresses assigned to the server and to listen on the port defined by the Port command.

LoginGraceTime Defines the amount of time the server will wait for the user to successfully complete a login. The default is 600 seconds.

LogLevel Specifies the level of detail written to the log. In rising level of detail, the possible values are QUIET, FATAL, ERROR, INFO, VERBOSE and DEBUG. The default is INFO.

MACs Identifies the message authentication code algorithms that will be used by the server. The default is hmac−md5, hmac−sha1, hmac−ripemd160, hmac−sha1−96, and hmac−md5−96.

MaxStartups Defines the maximum number of connections that can be awaiting authentication. The default is 10.

PAMAuthenticationViaKbdInt Specifies whether or not PAM challenge/response authentication is allowed. When specified, it will allow password authentication, even if PasswordAuthentication is disabled. The default is no.

PasswordAuthentication Tells sshd whether or not password authentication is allowed. By default, it is allowed.

PermitEmptyPasswords Tells sshd whether or not a password is *required* for password authentication. When set to yes, this parameter allows a user with an empty password to log in simply by pressing the carriage return in response to the password prompt. By default, this parameter is set to no. Therefore, a user must enter a password when password authentication is used.

PermitRootLogin Defines the level of access granted to the root user. Any one of four levels of access can be defined:

> **yes** The root user is given full login access through ssh. This is the default.
>
> **no** The root user is not allowed to access the system through ssh.
>
> **without−password** The root user is allowed to access the system, but cannot use password access to do so. A strong form of authentication, such as public key authentication, must be used.
>
> **forced−commands−only** The root user is allowed to remotely execute commands if the user has been authenticated by public key authentication. In no case will the root user be allowed to log in to a shell through ssh.

PidFile Defines the path of the file in which sshd writes its process ID. The default is /var/ run/sshd.pid.

Port Defines the port on which sshd listens for incoming traffic. The default is 22.

PrintLastLog Tells sshd whether or not it should log login activity in the last log. By default, it does.

PrintMotd Specifies whether or not sshd should display the message of the day when a user logs in. By default, it does.

Protocol Lists the SSH protocol versions that the server should support. By default, the server first tries version 2 and then falls back to version 1. Specified as a list on the Protocol command line, the default would be Protocol 2,1. Protocol 1 has some known security problems. It is a good idea to disable protocol 1 if you can.

PubkeyAuthentication Specifies whether or not public key authentication is allowed. By default it is.

ReverseMappingCheck Tells sshd to verify that DNS lists the same hostname and IP address pairing for the client in the reverse domain as was found when the hostname was resolved to an IP address. (See Chapter 4, "Linux Name Services," for a detailed description of the reverse domain.) This check can help detect DNS and address spoofing. However, it adds overhead to sshd, and it often rejects legitimate client systems. The default is no, which tells sshd not to attempt this check.

RhostsAuthentication Tells sshd that the highly insecure trusted host files .rhosts or /etc/ hosts.equiv are all that is needed for authentication. The default is no, and it should not be changed. Instead, use RhostsRSAAuthentication, along with .rhosts or /etc/hosts.equiv, for strong host–based authentication if you must use the trusted host files.

RhostsRSAAuthentication Tells sshd that it can use the trusted host files .rhosts and /etc/ hosts.equiv when it combines them with RSA host authentication. The default is no, which tells sshd not to use the trusted host files, even with strong authentication.

RSAAuthentication Specifies whether or not RSA public key authentication is allowed. By default, it is.

ServerKeyBits Defines the number of bits used for the session encryption for SSH version 1. The minimum value is 512 bits, and the default is 768 bits.

StrictModes Tells sshd to check the file and directory permissions and ownership before allowing a login. The default is yes, check the permissions.

Subsystem Defines a subsystem name and a command to execute when that subsystem is requested. The subsystem name is an external name available to the client. The command is a command that executes on the server. By default, no subsystems are defined.

SyslogFacility Identifies the syslogd facility that sshd should use to log messages. According to the documentation, this parameter will accept only DAEMON, USER, AUTH, or LOCAL0 through LOCAL7; and the default is AUTH. In reality, any facility defined in syslog.conf (and only those facilities) can be used.

UseLogin Specifies whether or not login is used for interactive login sessions. By default, it isn't.

X11DisplayOffset Defines the lowest display number that sshd is allowed to use for X11 forwarding. The default is 10.

X11Forwarding Tells sshd whether or not it should forward X11 traffic to the client. The default is no.

XAuthLocation Defines the path to the xauth program. The default is /usr/X11R6/bin/xauth.

Using the information from this list of parameters, you should be able to read the configuration delivered with your system, and modify it for your environment, if necessary. The /etc/ssh/sshd_ config file delivered with our sample Red Hat system is shown in Listing 12.11.

Listing 12.11: The Red Hat *sshd_config* file

```
#       $OpenBSD: sshd_config,v 1.38 2001/04/15 21:41:29 deraadt Exp $

# This sshd was compiled with PATH=/usr/bin:/bin:/usr/sbin:/sbin

# This is the sshd server system-wide configuration file.  See sshd(8)
# for more information.

Port 22
#Protocol 2,1
#ListenAddress 0.0.0.0
#ListenAddress ::
HostKey /etc/ssh/ssh_host_key
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
ServerKeyBits 768
LoginGraceTime 600
KeyRegenerationInterval 3600
PermitRootLogin yes
#
# Don't read ~/.rhosts and ~/.shosts files
IgnoreRhosts yes
# Uncomment if you don't trust ~/.ssh/known_hosts for
 ÄRhostsRSAAuthentication
#IgnoreUserKnownHosts yes
StrictModes yes
X11Forwarding yes
X11DisplayOffset 10
PrintMotd yes
#PrintLastLog no
KeepAlive yes

# Logging
SyslogFacility AUTHPRIV
LogLevel INFO
#obsoletes QuietMode and FascistLogging

RhostsAuthentication no
#
# For this to work you will also need host keys in
 Ä/etc/ssh/ssh_known_hosts
RhostsRSAAuthentication no
```

```
# similar for protocol version 2
HostbasedAuthentication no
#
RSAAuthentication yes

# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication yes
PermitEmptyPasswords no

# Uncomment to disable s/key passwords
#ChallengeResponseAuthentication no

# Uncomment to enable PAM keyboard-interactive authentication
# Warning: enabling this may bypass the setting of
 Ä'PasswordAuthentication'
#PAMAuthenticationViaKbdInt yes

# To change Kerberos options
#KerberosAuthentication no
#KerberosOrLocalPasswd yes
#AFSTokenPassing no
#KerberosTicketCleanup no

# Kerberos TGT Passing does only work with the AFS kaserver
#KerberosTgtPassing yes

#CheckMail yes
#UseLogin no

#MaxStartups 10:30:60
#Banner /etc/issue.net
#ReverseMappingCheck yes

Subsystem       sftp    /usr/libexec/openssh/sftp-server
```

Lines that begin with a hash mark (#) are comments. Some comments explain the use of a specific command. Some are commands that are commented out. These commented–out commands are useful as examples of the syntax. The meaning of the parameters in this file can be quickly understood by using the listing of commands provided earlier, but there are two interesting items worth noting.

First, notice that the file contains multiple HostKey statements. Each HostKey line points to a different key file, and each file contains a single key that is appropriate to a different public key encryption protocol. The ssh_host_key file holds the RSA key formatted for SSH version 1. The ssh_host_rsa_key file holds the RSA key for SSH version 2, and the ssh_host_dsa_key file holds the DSA key formatted for SSH version 2. The first time that sshd is started on a Red Hat system, the /etc/init.d/sshd script generates three keys, as shown below. These HostKey statements identify those key files to sshd.

```
[root]# service sshd start
Generating SSH1 RSA host key:                              [  OK  ]
Generating SSH2 RSA host key:                              [  OK  ]
Generating SSH2 DSA host key:                              [  OK  ]
Starting sshd:                                             [  OK  ]
```

Second, notice the Subsystem command at the end of the file, which provides a good example of how the Subsystem command is used. When the client requests sftp, the server runs sftp–server to provide the service. If this line is commented out of the sshd_config file, the secure FTP service sftp

is unavailable.

Configuring the *ssh* Client

The secure shell client, ssh, can be configured using the /etc/ssh/ssh_config file for system−wide configuration or the ~/.ssh/config file in the users home directory for user−specific configuration. The various client configuration options are listed in Table 12.2.

Table 12.2: ssh Client Configuration Options

Option	Usage
AFSTokenPassing	Specifies whether or not AFS tokens are passed to the server.
BatchMode	Disables the password prompt for batch mode execution.
CheckHostIP	Checks the host IP address in the known_hosts file to detect address spoofing.
Cipher	Specifies either blowfish or 3des as the session encryption used for SSH version 1.
Ciphers	Defines the acceptable session encryption techniques for SSH version 2.
Compression	Specifies whether or not compression should be used.
CompressionLevel	Identifies the level of compression to be used, ranging from 1 (fastest) to 9 (best).
ConnectionAttempts	Defines the maximum number of connection attempts before terminating.
EscapeChar	Defines the escape character. The default is a tilde (~).
FallBackToRsh	Tells ssh to try rsh if the ssh connection fails.
ForwardAgent	Specifies whether or not the connection to the authentication agent is forwarded to the remote machine.
ForwardX11	Specifies whether or not X11 connections are automatically redirected over the secure channel.
GatewayPort	Specifies whether or not remote hosts are allowed to connect to locally forwarded ports.
GlobalKnownHostsFile	Defines the path to the file that holds known host keys for SSH version 1. The default is /etc/ssh/ssh_known_hosts.
GlobalKnownHostsFile2	Defines the path to the file that holds known host keys for SSH version 2. The default is /etc/ssh/ssh_known_hosts2.
HostbasedAuthentication	Tells ssh that it can try public key host−based authentication.
HostKeyAlgorithms	Identifies the encryption techniques used for public keys and the order in which they should be used.
HostKeyAlias	Specifies an alias used to look up the key for a

	known host.
HostName	Specifies the real hostname of the remote host that ssh should log in to.
Host	Identifies the remote host to which the configuration commands are applied.
IdentityFile	Defines the pathname of the user's public key identification file.
KeepAlive	Specifies whether or not ssh should transmit keepalives.
KerberosAuthentication	Specifies whether or not Kerberos authentication is used.
Option	Usage
KerberosTgtPassing	Specifies whether or not Kerberos Ticking Granting Tickets (TGTs) are passed to the server.
LocalForward	Connects a local port to a port on the remote system via the secure channel.
LogLevel	Defines the level of detail that ssh should log.
MACs	Defines the message authentication algorithm that ssh will use.
NumberOfPasswordPrompts	Defines the maximum number of password attempts before the connection attempt is abandoned.
PasswordAuthentication	Specifies whether or not password authentication can be used.
Port	Defines the port number used to connect to the remote host. The default is 22.
PreferredAuthentications	Lists the order in which different authentication methods will be tried.
Protocol	List the order in which SSH versions will be tried.
ProxyCommand	Specifies a command that should be run to connect to the external server.
PubkeyAuthentication	Specifies whether or not public key authentication can be used.
RemoteForward	Connects a port on a remote machine to a local port.
RhostsAuthentication	Specifies whether or not ssh should try trusted host authentication.
RhostsRSAAuthentication	Specifies whether or not ssh should try trusted host authentication with public key authentication.
RSAAuthentication	Specifies whether or not ssh should try RSA authentication.
ChallengeResponseAuthentication	Specifies whether or not ssh should try challenge/response authentication.
StrictHostKeyChecking	Tells ssh not to automatically add new hosts to the known_hosts file. Set to yes, if you have a policy for distributing known_hosts files, to drastically limit the risk of a man−in−the−middle attack. Set to ask to allow the user to verify the remote host. no should not be used.

UsePrivilegedPort	Tells ssh to use a privileged port for outbound traffic.
User	Specifies the username for the login.
UserKnownHostsFile	Defines the path to the file that contains user keys for SSH version 1.
UserKnownHostsFile2	Defines the path to the file that contains user keys for SSH version 2.
UseRsh	Specifies whether or not ssh should use rsh and rlogin for this host.
XAuthLocation	Defines the path to the xauth program.

Many of the parameters in Table 12.2 are identical to parameters covered for sshd. The difference is that these parameters define these values for the client side of the connection. Listing 12.12 shows the ssh_config file from our sample Red Hat system.

Listing 12.12: The Red Hat *ssh_config* file

```
#       $OpenBSD: ssh_config,v 1.10 2001/04/03 21:19:38 todd Exp $

# This is ssh client systemwide configuration file.  See ssh(1) for more
# information.  This file provides defaults for users, and the values can
# be changed in per-user configuration files or on the command line.

# Configuration data is parsed as follows:
#  1. command line options
#  2. user-specific file
#  3. system-wide file
# Any configuration value is only changed the first time it is set.
# Thus, host-specific definitions should be at the beginning of the
# configuration file, and defaults at the end.

# Site-wide defaults for various options

# Host *
#   ForwardAgent no
#   ForwardX11 no
#   RhostsAuthentication no
#   RhostsRSAAuthentication yes
#   RSAAuthentication yes
#   PasswordAuthentication yes
#   FallBackToRsh no
#   UseRsh no
#   BatchMode no
#   CheckHostIP yes
#   StrictHostKeyChecking yes
#   IdentityFile ~/.ssh/identity
#   IdentityFile ~/.ssh/id_dsa
#   IdentityFile ~/.ssh/id_rsa
#   Port 22
#   Protocol 2,1
#   Cipher blowfish
#   EscapeChar ~
Host *
        ForwardX11 yes
```

The file shown in Listing 12.12 defines the system–wide defaults for the ssh client. All lines that begin with a hash mark (#) are comments. Most of them are there to show that the standard

defaults are being used and what those default values are. This file contains only two active lines:

> **Host *** The configuration commands in an ssh configuration file are always preceded by a Host statement, which identifies the remote host to which this set of commands applies. The ssh configuration file can have several Host statements, each followed by its own set of configuration commands. Listing 12.12 contains only one Host statement, and the configuration command that follows it applies to all remote hosts. The * on the Host statement is used to refer to all hosts.
>
> **ForwardX11 yes** This command tells ssh that it should allow X11 traffic over the secure connection.

Secure shell is an excellent way to have secure communications between two systems across the Internet. However, it does require that both systems have the secure shell software installed. When you control both ends of the link, this is not a problem. This provides a very good reason to take your Linux laptop with you when you travel.

Monitoring Your System

Monitoring your system is an essential part of security. It helps you discover what attacks are being launched against your system so that you can concentrate on plugging popular holes. Monitoring also lets you know when someone has successfully penetrated your defenses.

Some basic Linux commands can help you learn what constitutes normal activity on your system so that you know when things are out of the ordinary:

- Use the who command to find out who is logged in and what they're doing.
- Use the last command to find out when people normally log in.
- Use the log files, such as /var/log/secure, to monitor access to network services and to monitor failed login attempts.
- Use ps to find out what processes are normally running.
- Develop a feel for your system. Intruders often change that feel.

Use these commands to establish a feel for normal operation. Do not expect these commands to catch an intruder in the act. Be aware that if your system is broken into, all of these commands will probably be replaced with altered versions designed to hide illicit activity, and the log file will probably be stripped of incriminating information.

Security Monitoring Tools

In addition to using simple commands to learn about your system, you should use some of the tools that have been specifically designed to detect the holes that intruders exploit and the changes they make to your system. There are several, and many of them are available on the Internet.

TARA (Tiger Auditors Research Assistant) is an updated version of the venerable Tiger package. TARA, like Tiger, is a group of shell scripts and C programs that scan configuration files and filesystems looking for security problems. TARA is very easy to use. The source code for TARA is available from www–arc.com.

TARA is a security auditing tool that runs directly on the host system. There are also network–based security auditing tools that externally probe the system. SATAN (Security Administrator's Tool for

Analyzing Networks) was the first successful network–based security auditor. SATAN has had several follow–on products: SAINT (Security Administrator's Integrated Network Tool), SARA (Security Auditor's Research Assistant), and now Nessus. Nessus is available as both a source code distribution and an RPM from ftp://ftp.nessus.org/.

Another popular network–based auditor is Nmap. It is available in either source or RPM format from http://www.insecure.org/.

Pick one of the tools, run it, and use the report as a guide to possible security holes. Then, use your own judgment to determine how much of a threat the reported holes actually are. The security reports from the scanners are useful primarily as pointers to possible security problems. However, they can't replace your own judgment about what is right for your system.

Bad guys can use tools such as Nessus and Nmap to scan your network for vulnerabilities. PortSentry, part of the Abacus security software from Psionic, monitors your system's network ports to detect security scans. PortSentry gives you a chance to detect an attack before the intruder successfully penetrates your system. The PortSentry source code distribution is available on the web at http://www.psionic.com/. PortSentry is also available online in RPM format. These tools can help you detect an attack, but you must use your own judgment. Don't overreact until you know what is going on. There are many false alarms. These tools are meant to aid you, not replace your personal knowledge of what's going on.

A common complaint about security scanners is that they are out–of–date. It is very difficult for the security scanners to stay up–to–date because systems and problems change very rapidly. But even though these programs are dated, old security holes can be exploited just as effectively as new ones, so it is worth running a scanner at least once. Occasionally check the sites that distribute security scanners for updated tools. When new tools arrive, use them to recheck your systems.

In Sum

Attaching your server to a network increases the risk of security problems, and security problems are the greatest risk to the stability and reliability of your system. The most difficult task of running a network service is keeping the system up and running 24 hours a day, seven days a week. Linux makes that possible because it is rock solid. So solid, in fact, that whenever a Linux system crashes, I suspect an outside force, such as a security intruder.

However, security breaches are not the only threats to the reliability and stability of your Linux server. The next chapter describes the other kinds of network problems that can appear, and tells you what to do to troubleshoot those problems.

Chapter 13: Troubleshooting

Overview

It is better to avoid trouble than it is to fix things after trouble arises. Because avoiding trouble is one of the primary motivations for good security, some of the techniques described in Chapter 12, "Security" (such as keeping the system software and your knowledge of potential problems up–to–date) apply equally well, whether the threat is a security intruder or a bug that crashes your favorite application. The bug fixes posted at the vendor's site are not always about security, but they are almost always of interest to you.

There is a difference between fixing bugs and enhancing the system. Fix bugs that you have detected on your server or that you know are a direct threat to your server. Avoid installing things just to get a new feature—reliability is more important than new features for a server. Try out the latest bells and whistles on your desktop system or a test server, and debug them before you move them to the server.

Despite your best efforts, things *will* go wrong. It's inevitable. No matter how great your knowledge, you will make mistakes; and no matter how hard you try to avoid them, problems will appear. Networking your system only adds to this potential because mistakes made by someone far away can negatively impact your users.

This chapter examines several basic Linux commands that can help you analyze and solve network problems. But before getting to them, let's look at how to build an updated Linux kernel.

The heart of your Linux system is the kernel. Therefore, keeping the kernel updated is an essential part of keeping the system software up–to–date. This chapter begins by looking at how you can avoid trouble by keeping your Linux kernel current.

Configuring the Linux Kernel

The Linux kernel source code is included among the CD–ROMs of your Linux distribution. Updated kernel source code is available online from your Linux vendor. Additionally, the latest kernel source code can be obtained from ftp://ftp.kernel.org/ or http://www.kernel.org/.

The Linux kernel is a C program, compiled and installed by make. A make command creates the Makefile needed to compile the kernel for your system. The make command used to compile the Linux kernel accepts a few arguments that control the type of user interface used to configure the kernel.

- make config runs a text–based configuration interface.
- make menuconfig runs a curses–based configuration interface that displays menu selections on text–based terminals.
- make xconfig runs an X Window System configuration interface.
- make oldconfig builds the new kernel using the configuration from your previous kernel build.

This chapter uses make xconfig to customize the kernel configuration. There are a huge number of kernel–configuration options. make xconfig provides a nice X interface that allows you to go directly to those parts of the configuration that you want to modify. The ability to ignore those configuration options you don't need and jump directly to those you do need is very useful.

Configuring the Kernel with *xconfig*

To start the kernel configuration process, change to the /usr/src/linux directory, and run make xconfig, which opens the window shown in Figure 13.1.

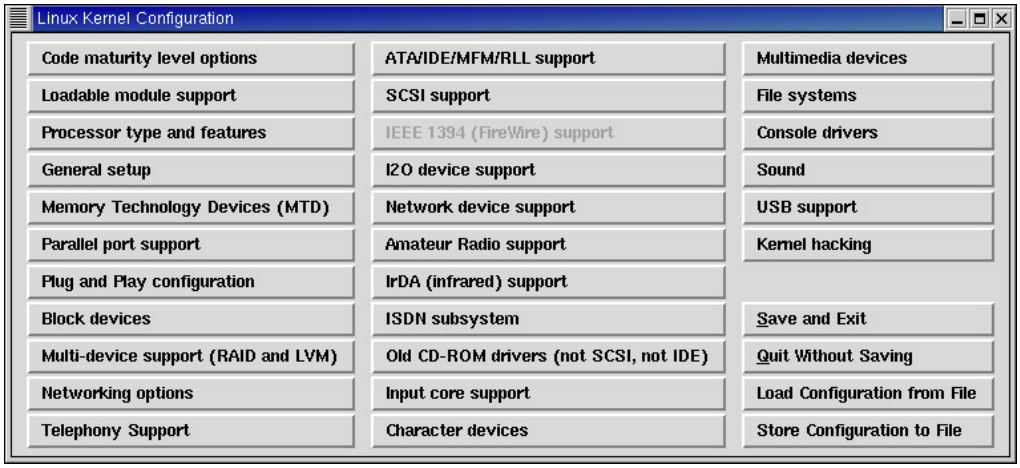


Figure 13.1: The Kernel Configuration window

The Kernel Configuration window displays more than 30 buttons that represent different configuration categories. (These buttons are described in detail in the upcoming "Understanding the Kernel Configuration Categories" section.) Click a button to view and set the configuration options in that category. Figure 13.2 shows the window that appears if the Network Device Support button is selected.

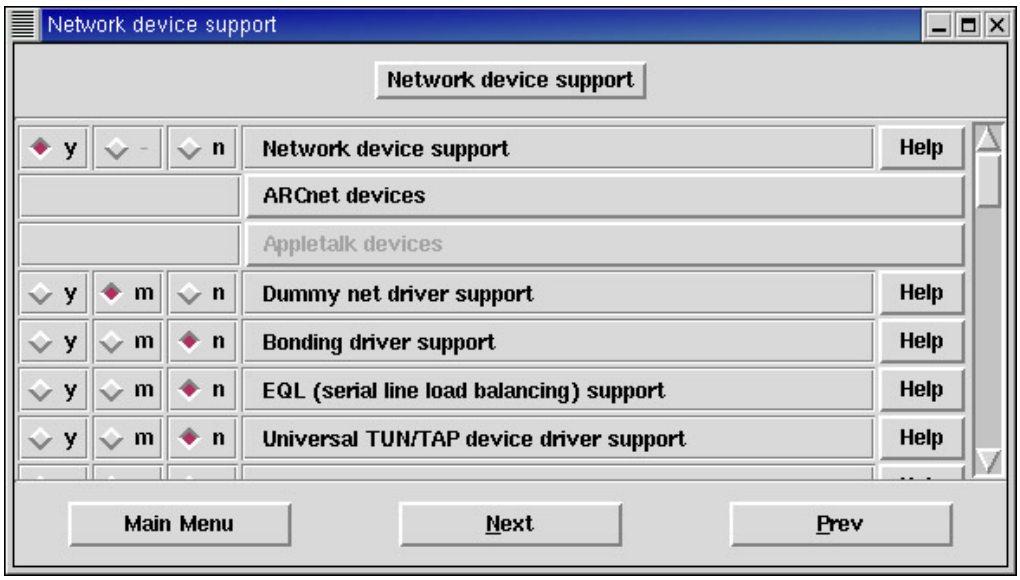


Figure 13.2: Network device support configuration options

The Network Device Support window lists all of the network device drivers that can be compiled into or loaded by the kernel. Many devices are listed directly in this window, and others are viewed by clicking on a button in this window. For example, Figure 13.2 shows an ARCnet devices button that produces a full listing of the available ARCnet devices. Scroll further down the Network Device Support window to find a similar button for Ethernet devices. For each network device, you have three standard choices, which are the same for most configuration options:

y Compiles the option into the new kernel.

m Causes the option to be built so it can be used as a loadable module by the kernel. Not every option is available as a loadable module. Notice the Network device support option in Figure 13.2. When a configuration question must be answered yes or no, the module selection is not available.

n Tells the kernel not to build the configuration option, although you can still build it as a module later.

In addition to these standard selections, some options offer a drop–down list of configuration keywords. For example, clicking the Pentium III button in the Processor type and features window shown in Figure 13.3 displays a drop–down list of processor types. Simply select the item you want from a drop–down list. Figure 13.3 shows this example of a drop–down list.

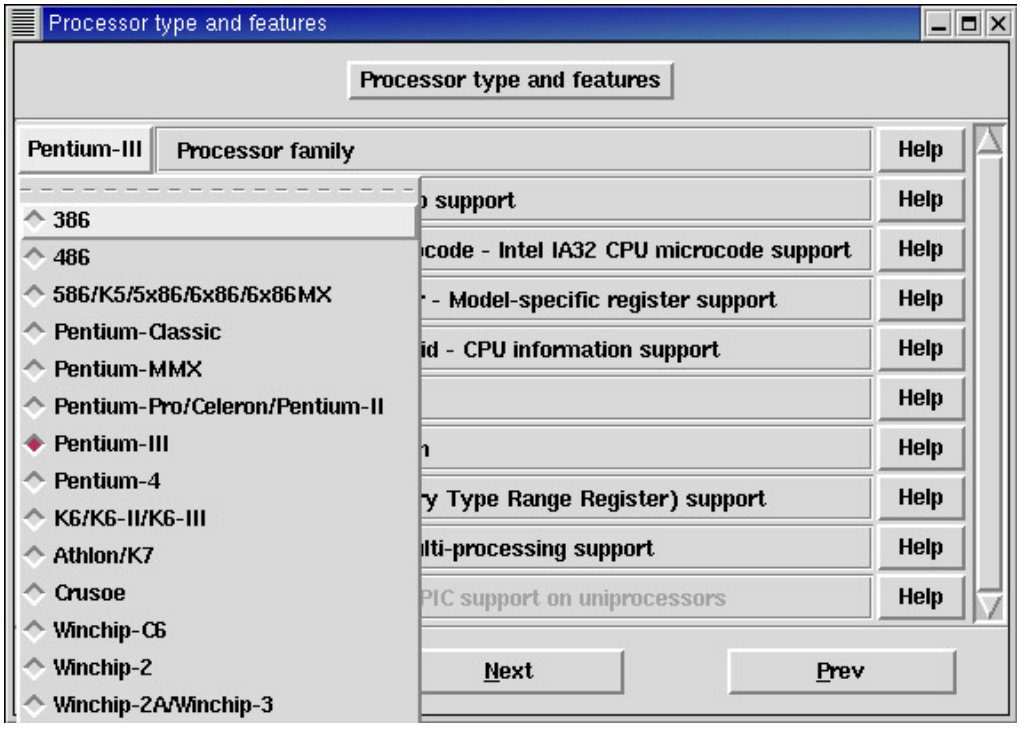


Figure 13.3: Selecting processor types and features

Each configuration option also has a Help button. Clicking the Help button provides additional information about the option and advice about when the option should be set. Even if you think you know what the option is about, you should read the description displayed by the Help button before you change any option from its default setting.

Understanding the Kernel Configuration Categories

Now that you know what is displayed when you select a configuration category, let's look at what the categories mean. More than half of the kernel configuration categories configure hardware subsystems and devices. This is not surprising given the kernel's role in managing hardware. The remaining categories cover options for configuring the kernel architecture and optional features, such as networking, which require kernel support.

The configuration categories shown in Figure 13.1 are the following:

Code maturity level options Specifies whether or not you want to include experimental code in the kernel.

Loadable module support The options in this category are used to enable support for loadable modules. Every option here should be set to Yes.

Processor type and features This category allows you to select the processor type, as described previously, and to set support for processor features, such as math emulation and memory type range register support. If you think you need them, you can say Yes to these features because they will not interfere with normal operation and will be used only if needed. Unless you have a server with more than one CPU, say No to symmetric multiprocessing support.

General setup This category has more than 25 different configuration options, including specifying support for networking, PCMCIA cards, and the PCI bus. Unless you have odd hardware or software requirements, the defaults should be correct.

Memory Technology Devices (MTDs) Enables support for Memory Technology Devices, such as flash RAM. Support for the Intel Common Flash Interface (CFI) is available through this category. Don't enable it unless you actually need it.

Parallel port support Enables support for the system parallel port.

Plug and Play configuration Specifies whether or not your system should support automatic configuration of Plug−and−Play devices. Common settings are to say Yes for Plug and Play.

Block devices Defines the block device support for the kernel. Block devices include hard disk drives and floppy drives. However, the devices listed here don't use the standard SCSI or IDE interfaces. The settings in this category can enable support for a wide range of unique block devices.

Networking options Defines the networking support for the kernel. There are more than 50 network configuration options. By default, the options are configured for a host. If you plan to run your system as a router, you should carefully examine the options that relate to the IP: Advanced Router option. Selecting that option provides several selections that tune the kernel for routing, as opposed to the normal state, which is to have the kernel tuned for a host computer.

Telephony support Enables support for a telephone card installed in the Linux system. Specific support for Quicknet Technologies telephony cards can be selected.

ATA/IDE/MFM/RLL support Enables support for the standard block device interfaces found on most PCs. Support for specific devices is enabled through the IDE, ATA, and ATAPI Block devices button found under this category.

SCSI support Enables support for SCSI devices. Low−level SCSI drivers, PCMCIA SCSI support, and support for SCSI tapes are also found in this category.

I2O device support Enables support for Intelligent Input/Output (I2O) devices. This is useful if the system has a independent I/O processor (IOP) installed on an I2O interface adapter.

Network device support Defines which network interface cards are supported. Most network interfaces are configured as loadable modules. This window was

shown in Figure 13.2.

Amateur radio support Defines whether or not the kernel should have support for an amateur radio network connection.

IrDA (infrared) support Defines whether or not the kernel should have support for the Infrared Data Association's infrared wireless network. If you use an infrared wireless network in your machine, enable the support here; otherwise, this should be set to No.

ISDN subsystem Defines the Integrated Services Digital Network (ISDN) support for the kernel. ISDN is an optional digital telephone service that can be purchased in many parts of the world. If you will be using an ISDN card on your system, you can configure these options.

Old CD−ROM drivers (not SCSI, not IDE) Specifies what nonstandard CD−ROM devices should be supported by the kernel. Operational servers do not use old nonstandard devices of any kind. Unless you use a nonstandard CD−ROM, you can say No to this option.

Input core support Enables support for USB Human Interface Devices (HID). When enabled, HID keyboard, mouse, and joystick support can be selected.

Character devices Defines which character devices are supported by the kernel. Character devices include terminals, serial ports, mice, joysticks, and floppy tape devices.

Multimedia devices Defines support for video−capture devices.

Filesystems Specifies the filesystem types that will be supported by the kernel. There are a large number of possible filesystems, including network−based filesystems, such as Coda, NFS, Novell NetWare, and Microsoft SMB protocols. Additionally, support for accessing partitions formatted by a wide range of operating systems is defined here.

Console drivers Defines the hardware support for the console monitor. It should specify the correct settings for your monitor and adapter card.

Sound Defines the hardware support for the sound card. Use this window to enable support for sound and to set the card configuration. Specific sound cards that are supported by the Linux kernel are defined here.

USB support Enables support for USB, and defines the support for a wide range of specific USB devices.

Kernel hacking Enables kernel debugging that allows certain keyboard sequences to control the kernel.

For a production system, limit the amount of experimental code in the kernel. Some experimental code can't be avoided. For example, IPv6 is considered experimental, but may be required at your site. However, it is best to avoid experimental code that you don't really need. You should also avoid any hardware you don't need. It is not always necessary, however, to say "no" to unneeded

hardware. If the hardware driver is a loadable module, it does not have to be loaded unless it is actually needed.

Compiling and Installing the Kernel

After selecting the configuration options you want, compile the kernel. To do so:

1. Run make dep ; make clean to build the dependencies and clean up the odds and ends.
2. Use make zImage to build a compressed kernel.
3. The loadable modules used by the kernel also need to be compiled, so run make modules to build the modules.
4. Run make modules_install to put the modules in the correct directory.
5. Because our sample Red Hat system uses an Intel processor, the newly compiled kernel is found in the /usr/src/linux/arch/i386/boot directory. The i386 path component would be something else (for example, sparc) on a system with a different hardware architecture. Copy the kernel, which in this example is stored under the name zImage, to the /boot directory. Give the copy of the kernel image file placed in /boot a unique name based on the kernel release number—for example, vmlinuz–2.4.18.

Booting the New Kernel

Modify the boot configuration to use the new kernel, while allowing you to fall back to the old kernel when necessary. Listing 13.1 shows a modified lilo.conf file that defines both the new and the old Linux kernels.

Listing 13.1: Adding the New Kernel to *lilo.conf*

```
[root]# cat lilo.conf
# global section
boot=/dev/hda3
prompt
timeout=50
message=/boot/message
default=linux
# the new boot image
image=/boot/vmlinuz-2.4.18
    label=linux
    read-only
    root=/dev/hda3
    password=Wats?Watt?
    restricted
# the old boot image
image=/boot/vmlinuz-2.4.2-2
    label=safe
    read-only
    root=/dev/hda3
    password=Wats?Watt?
    restricted
[root]# lilo
Added linux *
Added safe
```

Both boot images are defined in the lilo.conf file. The only differences in the definitions are the names of the image files and the label used by the operator to boot the system. By default, the system will attempt to boot the new kernel. If any problems are encountered, the operator can

reboot the system and enter the label safe at the boot prompt to boot the old kernel.

Listing 13.1 shows the configuration for a system that uses LILO. If you system uses GRUB, modify the grub.conf file to allow the operator to boot either one of the Linux kernels. Listing 13.2 shows a grub.conf file modified to boot either kernel.

Listing 13.2: Adding a New Kernel to *grub.conf*

```
[root]# cat /etc/grub.conf
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE:  You do not have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /, eg.
#          root (hd0,2)
#          kernel /boot/vmlinuz-version ro root=/dev/hda3
#          initrd /boot/initrd-version.img
#boot=/dev/hda
default=0
timeout=10
splashimage=(hd0,2)/boot/grub/splash.xpm.gz
password --md5 $1$LºÒCX≤Ëª$qgeIevUEDvvQAmrm4jCd31
title Linux (2.4.18)
        root (hd0,2)
        kernel /boot/vmlinuz-2.4.18 ro root=/dev/hda3
        initrd /boot/initrd-2.4.18.img
title Old Linux (2.4.7-10)
        root (hd0,2)
        kernel /boot/vmlinuz-2.4.7-10 ro root=/dev/hda3
        initrd /boot/initrd-2.4.7-10.img
```

By default, this configuration boots the new Linux kernel, but the menu displays both kernels to the operator. If anything goes wrong with the new kernel, the operator can fall back and boot the old kernel.

Note If you load a RAM disk image with the GRUB initrd option, you must build a RAM disk file for each kernel using the mkinitrd command.

The new kernel is now ready to run. Reboot the system, and put the new kernel in action. After you're comfortable with the new kernel, say after a month, you can remove the old kernel if you need the disk space.

Troubleshooting a Network Server

Even if you're liberal about fixing known bugs, and you're conservative about installing unneeded software, problems will occur. Many of these problems will be detected and reported by your users before you know anything is wrong. Tracking down those reported problems is as much an art as a science.

The art of troubleshooting is your intuition about the state of your server and the network, and your insight into the accuracy of the user's problem report. I don't say this to demean the intelligence of the user reporting the problem because I'm as guilty of providing inaccurate trouble reports as the next guy. When under stress, I have completely misunderstood very clear instructions and have

reported a problem when the only real problem was my lack of time to carefully read the instructions. Thus, you cannot assume too much from the trouble report, and you need to be methodical in applying your own knowledge to the problem. Here are some suggestions:

- Duplicate the problem yourself and then have the user duplicate the problem while you walk him through it. This often eliminates problems that spring from user confusion.
- Avoid oversimplification. The problem is not always a confused user. In a networked server, the problem can occur in any part of the network hardware or software, from your system to the remote system.
- Divide a complex problem into pieces and test the individual pieces to isolate the problem.

The science of troubleshooting is your knowledge of how your server and the network operate, and the tools that are available to conduct empirical tests of the server and the network. Your knowledge helps you focus on the proper area for testing, and helps you select the proper tools for testing that area. Here are some guidelines to help you make those decisions:

- If the problem occurs on outbound connections, it is probably unrelated to any of the network daemons running on your server. Use ifconfig to check the configur–ation of the network interface, ping to test the basic connectivity, and traceroute to test the route to the remote server. Talk with the administrator of the remote server to ensure that they are offering the service requested by the user and that it is properly configured.
- If the problem occurs on inbound connections, make sure that your system is running the required daemon. Connect to the daemon from the local host to make sure that it is running. If the daemon is running, ensure that it is properly configured. Check to make sure that the remote system is allowed access to the server by the enterprise firewall and by host–based access controls.
- If the problem occurs on only one client, concentrate on testing that client. If the problem happens for many clients, concentrate on the network and the server.

Diagnostic Tools

Troubleshooting may require both hardware and software tools. You may not think of yourself as a hardware person, but a small kit of hardware tools is useful to have on hand. A simple cable tester and a LAN maintenance toolkit (with every hand tool you'll ever need) can be bought at most places that sell LAN equipment. Add to those tools a dual–boot laptop, and you'll have all of the troubleshooting hardware you'll need.

A laptop is my favorite piece of test equipment. Configure the laptop to dual–boot Microsoft Windows and Linux. When a user reports a problem that you can't resolve sitting at your desk, take the laptop to the user's office, plug it directly into the Ethernet cable, and run the tests from there. This quickly tells you whether the problem exists in the user's computer or in the network.

Use a dual–boot system so that you have access to the great tools available on the Linux system and can demonstrate Microsoft Windows connectivity for the user. Some Microsoft Windows users don't believe the network is running unless you can show it running with Windows.

Linux includes a large array of simple software tools that can help you isolate a problem through testing. These software tools are sufficient to address most enterprise network problems. In addition to the application test tools (such as sendmail –bt that was covered earlier), simple troubleshooting tools included with Linux are as follows:

ifconfig Displays the network interface's configuration, characteristics, and

statistics.

arp and arpwatch Monitor the mapping of IP addresses to Ethernet addresses.

ping Tests basic network connectivity.

traceroute Tests routing and traces the network path end to end.

netstat Displays the status of ports and active connections.

nslookup, dig, and host Tests Domain Name Service. nslookup is a powerful interactive tool that comes with the BIND software. dig and host are similar tools that can be used inside of shell scripts.

tcpdump Analyzes network packets.

In the following sections, all of these software tools provided by Linux are examined.

Checking the Network Interface

Configuration errors, including those that allow security breaches, are often the cause of problems on mature systems. You cannot eliminate these problems simply by configuring your system correctly. When you're on a network, the configuration errors made by the administrator at the remote end of the network may affect your users. Additionally, you may be the expert called upon to help users correct the configuration errors they make when setting up their desktop systems.

Warning I specifically said "mature systems" because the desire to have the latest thing, even though it is not quite ready for prime time, can be an even bigger cause of problems. Beta software should be avoided for a production server unless its use is absolutely necessary.

Checking the configuration of a network server can mean reading configuration files as well as actively running tests. Red Hat Linux systems store the basic network interface configuration in files in /etc/sysconfig. To check the configuration of Ethernet interface eth0 on a Red Hat system, you could list the /etc/sysconfig/network file and the /etc/sysconfig/network–scripts/ifcfg–eth0 file, as shown in Listing 13.3.

Listing 13.3: Red Hat Network Interface Configuration Files

```
$ cat /etc/sysconfig/network
NETWORKING=yes
FORWARD_IPV4=false
HOSTNAME=parrot.foobirds.org
DOMAINNAME=foobirds.org
GATEWAY=172.16.12.254
GATEWAYDEV=eth0
$ cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
ONBOOT=yes
BOOTPROTO=
IPADDR=172.16.12.3
NETMASK=255.255.255.0
NETWORK=172.16.12.0
BROADCAST=172.16.12.255
USERCTL=no
```

```
TYPE=Ethernet
```

The arguments in these files are not the same on every Red Hat system. Many systems use DHCP, which is indicated in the ifcfg−eth0 file when BOOTPROTO=dhcp is set. If BOOTPROTO is set to dhcp, the DHCP server should provide the correct configuration. If it doesn't, the server configuration needs to be checked, and the network path from the client to the server needs to be checked. If a DHCP server is not used, the basic configuration values of hostname, IP address, network mask, network address, broadcast address, and default gateway should be set in the files shown in Listing 13.3. If the value assigned in each entry is correct, the configuration of the interface should be correct. Check that the configuration is in fact correctly defined by using the ifconfig command.

Note In Chapter 2, "The Network Interface," the ifconfig command was used to define the network configuration. Here, it is used to display the interface configuration.

Checking an Ethernet Interface

Enter the ifconfig command with the interface name and no other command−line arguments to display the current configuration. The example in Listing 13.4 shows the configuration of eth0 on our sample Red Hat Linux system.

Listing 13.4: Displaying the Configuration with *ifconfig*

```
$ ifconfig eth0
eth0   Link encap:Ethernet  HWaddr 00:00:C0:9A:72:CA
       inet addr:172.16.5.3  Bcast:172.16.5.255  Mask:255.255.255.0
       UP BROADCAST NOTRAILERS RUNNING  MTU:1500  Metric:1
       RX packets:22283 errors:0 dropped:0 overruns:0 frame:0
       TX packets:12321 errors:0 dropped:0 overruns:0 carrier:0
       collisions:26 txqueuelen:100
       RX bytes:32702095 (31.1 Mb)  TX bytes:841990 (822.2 Kb)
       Interrupt:5 Base address:0x250 Memory:c0000-c2000
```

The first line displayed by ifconfig lists the interface name (eth0), the link protocol (Ethernet), and the hardware address (00:00:C0:9A:72:CA). Of these, the first two are obvious, but the hardware address is a nice bit of information to have for tracking down physical layer network problems. (More on this when using arp to debug address assignment problems is discussed.)

The second line displays the IP configuration. It contains the IP address, the broadcast address, and the network mask assigned to the interface. These values should be exactly what you expect. As illustrated in Chapter 2, sometimes these values are not what you expect, which can be the cause of a network problem. Check these when the network problem seems to be localized to a single computer.

The third line of the display lists the flags that are set for this interface. The flags define the interface characteristics. In the example, the following values are set:

UP The interface is enabled. The interface should always be UP when the system is running. Use the ifconfig command to manually mark the interface DOWN only when changing a configuration value, such as the IP address. After making the change, re−enable the interface with the new configuration value by marking it UP. Here is an example of these steps:

```
# ifconfig eth0 down
# ifconfig eth0 172.16.5.9 up
```

BROADCAST The interface supports broadcasts. Some physical networks, such as Ethernet, provide built–in support for broadcasting packets.

NOTRAILERS The interface does not use trailer encapsulation. Trailer encapsulation was a techniques used to speed the processing of Ethernet packets on early networked computers. Trailer encapsulation is not used on modern systems.

RUNNING The interface is operational. The interface must be RUNNING at all times. If it isn't, the driver for this interface may not be properly installed.

MTU The maximum transmission unit used by this interface. In the example, it is 1500 bytes, which is the standard for Ethernet.

Metric The routing cost associated with this interface. In the example, it is 1, which is the normal cost associated with all network interfaces.

One other flag that you might see if you run the tcpdump tool on your system is the following:

PROMISC The interface is set to promiscuous mode. Normally, an Ethernet interface only passes packets that contain the local host's Ethernet address or the Ethernet broadcast address up to the IP layer for processing. Promiscuous mode means that the interface accepts all packets and sends them all up to IP for processing. This feature is required by the tcpdump tool, which is covered later in the chapter. If you don't intend to run tcpdump, this feature should not be used. To turn it off, enter the following ifconfig commands:

```
# ifconfig eth0 down
# ifconfig eth0 -promisc up
```

The next three lines of output from the ifconfig command that are shown in Listing 13.4 contain network statistics. The first of these lines provides statistics for received packets:

```
RX packets:22283 errors:0 dropped:0 overruns:0 frame:0
```

This line provides the total number of packets received, the number of received packets that contained an error, the number of packets dropped by the interface, the number of times an input buffer overrun occurred, and the number of framing error occurrences. A high percentage of drops and overruns can indicate that your server is so overloaded with work that it cannot properly service the input packet buffer. A high percentage of framing errors and corrupted packets (errors) can indicate a physical network problem. In the example, no receive errors were encountered. If all of the error counts are low, as they are in this example, physical network problems are not indicated.

The other two lines of statistics relate to the transmitted packets:

```
TX packets:12321 errors:0 dropped:0 overruns:0 carrier:0
collisions:26 txqueuelen:100
```

The first of these lines starts with the total number of packets transmitted. Displayed next are the number of transmitted packets that contained an error, the number of packets dropped by the interface, the number of times an output buffer overrun occurred, and the number of times the carrier signal was unavailable for a transmission. In the example, no output errors occurred. The

second line contains two fields. The txquelen field is not really a statistic; it simply displays the maximum size of the transmit queue. The collisions field indicates the number of times a transmission resulted in a collision. The example has a very low number of collisions, but a high number in the collision field is not unusual on a shared Ethernet.

Ethernet uses a technique called Carrier Sense Multiple Access with Collision Detection (CSMACD) to share the cable among many hosts. The best model for understanding how CSMACD works is a human conversation. To speak during a lively conversation, you wait for a lull in the discussion—that lull is the human equivalent to a carrier signal. When you sense the lull, you begin to speak—that's the transmission. If someone begins speaking just before you do, you hear him speaking and you stop—that's the collision detection. After giving the person his say, you speak again.

In CSMACD, a collision is when two or more Ethernet nodes attempt to transmit at the same time. Collisions become a problem only when there is so much traffic on a shared network that the collisions inhibit throughput. The collision rate is determined by calculating the collisions as a percentage of the total packets transmitted. For example, suppose that 35,531 collisions occurred for 1,041,083 transmissions, which gives a collision rate of about three percent. That's a heavily used network. If the collision rate reaches five percent, you should segment the network by moving from shared hubs to Ethernet switches. Allocating dedicated switch ports to your busy servers should dramatically reduce the collision rate.

The last line displayed by ifconfig in Listing 13.4 provides the configuration settings used for the Ethernet adapter card. These values must match the configuration actually set on the adapter card. See the insmod example in the "Configuring an Ethernet Device Driver" section of Chapter 2 for information on how the correct adapter configuration is passed to an Ethernet device driver. If you're new to the PC hardware used by Linux systems, see the "Adapter Card Configuration" sidebar.

Adapter Card Configuration

New PC cards are better than old ISA cards, not just because they are faster, but because they are easier to configure. For example, PCI adapters can share interrupts without a conflict. ISA cards cannot share configuration values, or else I/O conflicts can occur. ISA PC adapter cards require up to four distinct hardware configuration parameters: the Interrupt Request number (IRQ), the Direct Memory Access (DMA) Request number (DRQ), the I/O port address, and the adapter memory address. The default configuration values are often correct, and the best ISA cards have intelligent software configuration programs that help you avoid setting the wrong configuration value. Nonetheless, correct hardware settings are required for the adapter to function correctly. Each card must have unique configuration values, and each parameter has a limited range of possible values:

- The IRQ values available for adapter cards are 2–7, 9–12, and 15. IRQs 0, 1, 8, 13, and 14 are used by functions on the system board; and some others are used by standard hardware interfaces.
- The available DRQ values are 0–3 and 5–7. DRQ 4 is used by the system board.
- The I/O port addresses available for peripheral bus I/O are hex values in the range from 100 to 3FF. The I/O addresses from 000 to 0FF are reserved for the system board, and others are used by common hardware.
- The adapter memory address is the address in system memory in which the ROM on the adapter card is mapped. The addresses available for adapter memory are from C0000 to DFFFF.

Without an understanding of the four basic adapter configuration values, the adapter configuration

information displayed by ifconfig is meaningless. With such an understanding, this message provides useful information that can help you troubleshoot a hardware configuration error.

If you must troubleshoot an ISA adapter configuration, don't shy away from examining the configuration that Microsoft Windows has detected for this device. When the vendors of these old ISA cards created the cards, Linux was not yet well–known. The best tools for these old cards often exist only in a Windows version.

Resolving Address Conflicts

Most common interface configuration problems are easy to detect because they cause a hard failure. For example, a bad subnet mask causes a failure every time the user attempts to contact a system on another subnet.

However, a bad IP address can be a more subtle problem. If the IP address is grossly misconfigured (that is, if the network portion of the address is incorrect), the system will have a hard failure that is easy to detect. But if the host portion of the address is incorrect, the problem may go undetected for a long period of time.

Let me explain. Assume that a PC user is assigned the address 172.16.12.13, but accidentally enters the address as 172.16.12.31. This is a valid address; it's just not the correct one for the PC. Everything works fine until the Linux system that is really assigned 172.16.12.31 comes online, which might be a long time on a small network. After the real owner of the address joins the network, problems emerge, but they are not necessarily hard failures, and they may not even affect the PC that misappropriated the address. The problems are intermittent because sometimes the PC answers the ARP request first, and sometimes the Linux system answers first. Therefore, some computers have the PC mapped to the address in their ARP tables, whereas others have the Linux system in their tables. To further complicate matters, as ARP table entries time out and are remapped, the mix of which computers have which system mapped to the address constantly changes.

Using *arp* to Check for Address Conflicts

The arp command can be used to view and, if necessary, to change the contents of the ARP table. An ARP problem is indicated when the wrong host responds to a request or when an error message about duplicate IP addresses is displayed. If you suspect a problem with the ARP table, display the contents of the table immediately after the problem occurs. Act quickly because the ARP table is refreshed every few minutes, so the evidence of the problem may be lost.

The –a command–line argument displays the hostname, IP address, and Ethernet address of every system listed in the ARP table. If arp cannot resolve the IP address from the ARP table into a hostname, a question mark is displayed in the hostname field. The arp –a command also displays the Ethernet interface through which the information was learned. Listing 13.5 was produced on our sample Red Hat Linux system using the arp –a command.

Listing 13.5: Viewing the ARP Table

```
$ arp -a
is1 (172.16.55.251) at 08:00:20:82:D5:1D [ether] on eth0
dog (172.16.55.178) at 08:00:20:9A:4F:25 [ether] on eth0
warthog (172.16.55.33) at 08:00:20:87:B7:6E [ether] on eth0
```

```
fs1 (172.16.55.250) at 08:00:20:8D:19:7B [ether] on eth0
sloth (172.16.55.36) at 08:00:20:71:97:06 [ether] on eth0
crow (172.16.55.183) at 08:00:20:82:DA:43 [ether] on eth0
snipe (172.16.55.1) at 0A:00:20:18:48:31 [ether] on eth0
duck (172.16.55.110) at 08:00:5A:09:C3:46 [ether] on eth0
? (172.16.55.216) at 08:00:4E:34:70:92 [ether] on eth0
rabbit (172.16.55.181) at 00:20:AF:16:95:B0 [ether] on eth0
deer (172.16.55.145) at 08:00:69:0A:47:2E [ether] on eth0
```

If you know the Ethernet addresses of the hosts on your network and the IP addresses that are supposed to be assigned to those hosts, it is easy to detect the error in this list. If you don't have that information, you need to do some research.

Suppose that rabbit is the system you're having problems communicating with, and that you suspect an ARP table problem. You can focus on that single host and ignore the other entries in the table. From the console of rabbit, it is easy enough to use ifconfig to obtain the correct Ethernet address. You can then delete the bad ARP entry and place the correct Ethernet address in the client's ARP table, as follows:

```
# arp -d rabbit
# arp -s rabbit 00:00:C0:DD:DA:B1
```

This temporarily fixes the problem for one client, but it is not a solution. The real problem is locating the culprit at 00:20:AF:16:95:B0 that is passing itself off as rabbit. Without a list of the Ethernet addresses on your network, you must enlist the help of your users to check the Ethernet addresses of their computers looking for the address in question. Clearly, having a map of the IP addresses and Ethernet addresses on your network is important. arpwatch can help you build that map.

Building an Ethernet List with *arpwatch*

The arpwatch command is included with our sample Red Hat system. arpwatch monitors ARP activity, dynamically builds a map of Ethernet and IP address assignments, and sends informative messages about ARP to the root user. The dynamically constructed database of Ethernet/IP address mappings is stored in arp.dat. On Red Hat systems, that file is found in /var/arpwatch. An excerpt from an arp.dat file is shown in Listing 13.6.

Listing 13.6: The *arpwatch arp.dat* File

```
$ head -15 /var/arpwatch/arp.dat
8:0:20:22:fd:51  172.16.55.59     928927150       dopey
0:0:c0:9a:d0:db  172.16.55.59     928894146       dopey
0:0:c:43:8d:fb   172.16.55.254    930939507
0:e0:29:0:be:b9  172.16.55.173    930939632       goat
0:60:97:5b:69:62 172.16.55.182    930939614       herbivore
8:0:20:82:da:43  172.16.55.183    930939633       crow
0:e0:29:0:bd:93  172.16.55.19     930939652       dingo
0:10:4b:87:de:49 172.16.55.119    930865112       dot
0:10:4b:87:e0:77 172.16.55.34     930937149       owl
0:aa:0:a3:55:cb  172.16.55.34     929662402       owl
0:a0:24:8d:ea:d1 172.16.55.26     929509418       donkey
0:a0:24:d6:26:c0 172.16.55.185    930938674       shrike
0:e0:29:5:8e:83  172.16.55.29     930939527       pike
8:0:20:71:97:6   172.16.55.36     930939649       sloth
a:0:20:18:48:31  172.16.55.1      930939610       snipe
```

Entries in the arp.dat file contain four fields: the Ethernet address, the IP address, a time stamp, and the hostname. If the hostname cannot be determined, the fourth field is blank—as it is in one line of the sample file.

Simply looking at this file can tell you some interesting things. First and foremost, it is good to know what Ethernet address goes with what IP address. As you saw earlier, this can be useful for debugging certain types of address problems. Second, a quick look at this file can tell you if possible address conflicts already exist. Look at the entries for dopey and owl in the sample file. These entries show two different Ethernet addresses attempting to use the same IP address. I'm dopey, so I know the reason that two systems have used that IP address is because I move dopey from one hardware platform to another. However, I don't know why two Ethernet cards have duplicated the address of owl. That situation needs to be watched.

Keeping a Watch on *arp*

You don't have to read the arp.dat file to find out when new Ethernet–to–IP–address mappings are added or to find out if two Ethernet addresses are using the same IP address. arpwatch monitors ARP activity and automatically e–mails reports to the root user when something significant occurs. arpwatch mails the following reports:

Changed ethernet address The IP address is using a different Ethernet address than the one previously stored in the arp.dat database. This can simply mean that the IP address has been legitimately moved to a new system or that a new Ethernet card has replaced the old card in the system. However, if combined with other evidence (see the next message type), this can indicate a duplicate IP address problem.

Flip flop The IP address has reverted back to the Ethernet address it was using previously. Multiple flip flop messages clearly indicate that a single IP address is moving between two different Ethernet addresses. Unless you personally know the cause, investigate flip flop messages.

New activity An old Ethernet/IP address pairing that has been unused for six months or more is now back in use. I have never received this message; on my network, nothing sits unused for six months.

New station A new Ethernet/IP address pairing has just been detected. In an earlier example, I pointed out an error from a user mistakenly entering 172.16.12.31 instead of 172.16.12.13. As soon as that system came online, the following e–mail message would be sent to the root user account of the server:

```
From: arpwatch@wren.foobirds.org (Arpwatch)
To: root@wren.foobirds.org
Subject: new station (?)

            hostname: ?
          ip address: 172.16.12.31
    ethernet address: 0:10:4b:87:f7:e1
     ethernet vendor: 3Com 3C905-TX PCI
           timestamp: Friday, May 17, 2002 19:33:22
```

If you had just given a user the address 172.16.121.13 and received this e–mail message, the user's error would be obvious. Notice that arpwatch was unable to provide a hostname in this message; this is because 172.16.12.31 does not have a name assigned yet. If the user had used

the correct address, the name associated with that address would have appeared in the message.

In the sample arp.dat file shown in Listing 13.6, there are two Ethernet addresses assigned to owl. That duplicate address assignment produced the three e−mail messages shown in Listing 13.7.

Listing 13.7: Sample *arpwatch* E−mail Reports

```
Subject:changed ethernet address (owl.foobirds.org)
   Date:Thu, 16 May 2002 03:41:31 -0400
   From:arpwatch@wren.foobirds.org (Arpwatch)
     To:root@wren.foobirds.org

            hostname: owl.foobirds.org
          ip address: 172.16.55.34
    ethernet address: 0:10:4b:87:e0:77
     ethernet vendor: 3Com 3C905-TX PCI
old ethernet address: 0:aa:0:a3:55:cb
 old ethernet vendor: Intel
           timestamp: Thursday, May 16, 2002 3:41:31
  previous timestamp: Thursday, May 16, 2002 2:58:21
               delta: 43 minutes

Subject:flip flop (owl.foobirds.org)
   Date:Thu, 16 May 2002 19:33:22 -0400
   From:arpwatch@wren.foobirds.org (Arpwatch)
     To:root@wren.foobirds.org

            hostname: owl.foobirds.org
          ip address: 172.16.55.34
    ethernet address: 0:aa:0:a3:55:cb
     ethernet vendor: Intel
old ethernet address: 0:10:4b:87:e0:77
 old ethernet vendor: 3Com 3C905-TX PCI
           timestamp: Thursday, May 16, 2002 19:33:22
  previous timestamp: Thursday, May 16, 2002 19:31:19
               delta: 2 minutes

Subject:flip flop (owl.foobirds.org)
   Date:Thu, 16 May 2002 19:33:22 -0400
   From:arpwatch@wren.foobirds.org (Arpwatch)
     To:root@wren.foobirds.org

            hostname: owl.foobirds.org
          ip address: 172.16.55.34
    ethernet address: 0:10:4b:87:e0:77
     ethernet vendor: 3Com 3C905-TX PCI
old ethernet address: 0:aa:0:a3:55:cb
 old ethernet vendor: Intel
           timestamp: Thursday, May 16, 2002 19:33:22
  previous timestamp: Thursday, May 16, 2002 19:33:22
               delta: 0 seconds
```

The changed Ethernet address message shows when the new Ethernet address first became associated with owl. This is followed by two flip flop messages, showing that owl is alternating between two Ethernet addresses. The second flip flop message has a delta of zero seconds. (The delta tells you how much time elapsed between the change of Ethernet addresses.) A delta of zero seconds indicates that two systems with two different Ethernet addresses responded to the same ARP request. This tells you that you definitely have a duplicate address problem.

arpwatch is helpful for resolving duplicate address problems, and is useful for documenting the Ethernet/IP address mappings on your network. However, Ethernet is not the only network over which TCP/IP runs. Occasionally, you may also be called upon to debug a PPP link running over a modem connection.

Checking a PPP Interface

Troubleshooting a PPP connection can be complex because of the added layers of hardware and software involved. In addition to the TCP/IP software, the connection uses PPP software and a scripting language, such as chat or dip, to establish the connection. The hardware uses a serial port, a serial device driver, and an external modem—which also has its own command language. To fully test a PPP connection, you need to check all of these things.

Chapter 2 describes PPP configuration and the design of login scripts. Both chat and dip can be run with a –v option to monitor the progress of the script. When used with dip, –v echoes each line of the script to the controlling terminal as it is executed. When used with chat, –v sends the script errors to syslogd. To monitor the execution of a chat script in real time, use the –V option and the pppd –detach option. For example:

```
# pppd /dev/ttyS1 56700 connect "chat -V -f my-script" \
       -detach crtscts modem defaultroute
```

The –V option associated with the chat command sends a line–by–line execution trace of the script to stderr. The –detach parameter associated with the pppd command leaves the controlling terminal attached, which means that stderr will be displayed on the terminal. Thus, you can watch and debug your chat script in real time.

Testing the Link Hardware

To test a PPP link, separate the hardware from the software by testing the port, the driver, and the modem with a tool that doesn't depend on PPP. Any terminal emulator will do the job. One emulator that is available on most systems is minicom.

As the root user, run minicom with the –s option to display the configuration menu. Use the menu to make sure everything is configured the way it should be to test the modem and the link. There are several selections in the mincom configuration menu, but for the purpose of testing the PPP link hardware, only the Serial Port Setup selection is meaningful. Many small Linux systems have only one modem, and the serial port to which that modem is attached is often linked to the device name /dev/modem. A PPP server, however, may have several modems, so they are usually referenced by the names of the serial port through which they are connected (for example, /dev/cua0). Use the Serial Port Setup menu selection to ensure that minicom is configured to use the same port at the same speed as the specific modem you wish to test. It wouldn't help much to test the wrong serial port!

After minicom is configured, entering the minicom command should connect you directly to the modem. From there, you can use modem commands to dial the telephone number and directly connect to the PPP server at the remote location. Listing 13.8 shows a sample test using minicom.

Listing 13.8: Testing a PPP Link with *minicom*

```
Welcome to minicom 1.83.1

OPTIONS: History Buffer, F-key Macros, Search History Buffer, I18n
```

```
Compiled on Aug 28 2001, 15:09:33.

Press CTRL-A Z for help on special keys

AT S7=45 S0=0 L1 V1 X4 &c1 E1 Q0
OK
atz
OK
atdt3015551234
CONNECT 28800 V42bis
^M
Enter username> craig
Enter user password> Watts?Watt?
            Welcome to the PPP Modem Pool

PORT-2> show us
20 May 2002  09:05:44
Port    Username             Status          Service

  1     dave                 PPP
  2     craig                Executing Cmd

PORT-2> logout

Xyplex Logged out port 2 at 20 May 2002  09:06:48
NO CARRIER
^A


CTRL-A Z for help | 38400 8N1 | NOR | Minicom 1.83.1   | VT102 |      Offline
X
```

This example contains lots of output from minicom, the modem, and the remote server. Mixed in with that is a little bit of user input, which is indicated by bold type. The first thing in Listing 13.8 is welcoming information from minicom, followed by a modem initialization command (AT) issued by minicom. The response (OK) to that command tells you that the modem is operational.

Ever doubtful, we issue our own modem reset command (atz), to which the modem responds. Next, the atdt command is used to dial the telephone number of the remote PPP server. The server answers. We log in, and run a command. When convinced that everything is running fine, we log out of the server and close minicom by typing Ctrl+A, followed by x. This test tells us the following information:

- The correct port is configured.
- The port is operational.
- The correct line speed is set.
- The modem is operational.
- The phone line is operational.
- The remote modem is correctly configured.
- The remote server is operational.
- The login is correct for the remote server.

Failures in a terminal emulator test focuses further testing on these possible PPP problem areas:

- If the modem fails to respond to commands, the problem must be somewhere between your system and the modem. You should ensure that the port is correct and defined in /dev, the modem cable works, and the modem itself works.

- If the local modem works but it is unable to connect to the remote modem, double–check the phone number, and check with the remote system administrator to make sure that the modems are compatibly configured.
- If the modems display a "connect" message but the remote server does not respond, the configuration problem is either between the remote system and its modem or between the two end systems. Again, call the remote system administrator and check the configuration.
- If the login fails, double–check the username and password on the remote server.
- If all of this succeeds and the PPP connection is still not made, the problem lies in the PPP configuration of the two end systems. It is possible that the connection is failing because of parameter disagreements or because the authentication is failing. Again, this must be worked out with the remote system administrator.

The next section of this chapter moves the testing up from the physical interface. After the interface is debugged and operational, the focus of testing is on the flow of IP datagrams across the network.

Testing the Connection

The ping tool is used to test the network connection between your system and some remote host. ping is both simple and powerful. Its power lies in the fact that it does not depend on the application configuration of either end system. ping uses the ICMP Echo message, which tests the connection from the local IP layer to the remote IP layer. A ping test separates the network from the application. Any tool that helps you divide a network problem into separate pieces so that those pieces can be tested individually is extremely useful.

The Message of a Successful *ping*

A successful ping test tells you that the remote system is reachable. This means that the interface configuration of both the local and the remote system must be correct, the routing configuration of both systems must be correct, and the network hardware must be operational end to end. A successful test eliminates a lot of potential problems, allowing you to concentrate your testing.

For example, assume that a user reported that she was unable to use ftp to connect to dog.example.org. You could run a ping test. A successful ping test is shown in Listing 13.9.

Listing 13.9: A Successful *ping* Test

```
$ ping dog.example.org
PING dog.example.org (172.32.30.2): 56 data bytes
64 bytes from 172.32.30.2: icmp_seq=0 ttl=32 time=1.0 ms
64 bytes from 172.32.30.2: icmp_seq=1 ttl=32 time=0.7 ms
64 bytes from 172.32.30.2: icmp_seq=2 ttl=32 time=0.7 ms
64 bytes from 172.32.30.2: icmp_seq=3 ttl=32 time=0.7 ms
^C
--- 172.32.30.2 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.7/0.7/1.0 ms
```

This test shows that the connection to dog.example.org is running perfectly. The summary line says that every packet transmitted was successfully echoed back, and none was lost. The individual lines report information about each packet. The ttl field prints the TCP Time–To–Live value. It is not significant here, but we will see later how the traceroute command makes use of this field. The

sequence numbers (icmp_seq) tell you that every packet was received in sequence, and the round–trip times tell you that you have a fast connection to the remote site. A high packet loss, packets arriving out of sequence, or high round–trip times can indicate a congested network or a bad connection. But none of those things is happening in this example. Clearly, the user's problem lies elsewhere.

After verifying that the user's report about not being able to log in with ftp is true, check with the remote system administrator to see if they allow ftp access and are actually running the ftp daemon. Sometimes, services are blocked at the server or at the firewall for security reasons, and the user does not know it.

The Message of a Failed *ping*

A failed ping test can also tell you a lot. Listing 13.10 shows a ping test failure.

Listing 13.10: A Failed *ping* Test

```
$ ping 172.16.2.2
PING 172.16.2.2 (172.16.2.2): 56 data bytes
ping: sendto: Network is unreachable
ping: wrote 172.16.2.2 64 chars, ret=-1
ping: sendto: Network is unreachable
^C
--- 172.16.2.2 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
```

Again, the test directs you to focus your troubleshooting efforts on certain layers of the network. A failure indicates you should focus on the network hardware, interface configuration, and lower–layer network services. The error message further helps refine that focus.

The specific text of ping error messages varies slightly, but the messages fall into three main categories:

Network unreachable This indicates that the local host does not have a valid route to the remote computer. (The next section looks at some tools that can help you try to track down a routing problem.) A related message that you might see in a ping test is ICMP Redirect, which is not really an error. An ICMP Redirect means that you have the wrong route in your routing table for this destination, and the local router is correcting your routing table.

No answer This indicates that the remote host did not respond to the ICMP Echo packets. On some systems, the Linux ping command does not display an error message saying No answer. Instead, the error "host unreachable" is displayed. In either case, however, when the ping is terminated, the summary line says 100% packet loss, which means that the remote host did not respond. No answer can be caused by many things. Any interruption of service anywhere on the network, from your host to the remote system, can cause this problem. This error means that no IP packets can successfully travel from your host to the remote host. Look for network errors.

Unknown host This indicates that name service was not able to resolve the hostname into an address. Possibly the user gave you the wrong hostname or the

DNS is misconfigured. Later in the chapter, tools for testing DNS servers are discussed.

ping is one of the best test tools available to the system administrator. Unfortunately, some sites block ICMP Echo messages at their firewalls or drop them at their routers. Every system that offers services to external users should respond to ping, even if those responses really come from a dedicated ping server sitting on the same network as the system that provides external services. The Internet works best when everyone works together. Unfortunately, bad people have frightened some administrators so much that we are losing some good tools. Interfering with ICMP Echo limits the effectiveness of one of the simplest and best test tools.

The results of the ping test guide you in the next phase of testing. ping directs you to focus on routing, DNS, or the application as the cause of the problem. Linux provides tools to test all of these things. The next section discusses the tools that let you test routing.

Testing Routing

When a routing problem is indicated, the first thing to do is examine the routing table to make sure that the necessary routes for the local interface and the default route are defined. Use the route command with the –n option to display the table, as shown in Listing 13.11.

Listing 13.11: Displaying the Routing Table

```
$ route -n
Kernel IP routing table
Destination Gateway     Genmask         Flags Metric Ref Use Iface
172.16.12.3 0.0.0.0     255.255.255.255 UH    0      0     0 eth0
172.16.12.0 0.0.0.0     255.255.255.0   U     0      0     0 eth0
127.0.0.0   0.0.0.0     255.0.0.0       U     0      0     0 lo
0.0.0.0     172.16.12.2 0.0.0.0         UG    0      0     0 eth0
```

The various fields in the routing table listing are described in Chapter 7, "Network Gateway Services." For this check, however, the details aren't important. You just want to make sure that the route to the remote host exists in your routing table—either a specific route to the network that the host is attached to or a default route. (The default route has a destination of 0.0.0.0.) If you built the routing table from static routing entries, this will probably be a default route. If the table was built dynamically by a routing protocol, it might be very large and contain a specific route for the remote network. Regardless of how it got there, you need to make sure that your system has the necessary route.

Note In the case of a large routing table, use grep with the route command to search for a default route or a specific route.

Using *traceroute*

When you're sure that your system has the proper routes, use the traceroute command to test the route end to end. traceroute traces the route of UDP packets or ICMP echo packets through the network, and lists every router between your computer and the remote hosts. It does this by sending out UDP packets with small time–to–live (TTL) values and invalid port numbers to force ICMP errors and to record the sources of those errors.

Here's how it works. The TTL field is intended to ensure that packets do not loop through the network forever. Every router that handles a datagram subtracts 1 from the datagram's TTL. If the TTL reaches 0, the router discards the datagram and sends an ICMP Time Exceeded error message back to the source of the packet. Normally, the TTL is set to 255 when a datagram is created. This guarantees that the datagram can reach any point in the Internet while preventing the packet from circling the world forever.

traceroute sends out three UDP packets with a TTL of 1, followed by three packets with a TTL of 2, followed by three packets with a TTL of 3, and so on up to a TTL of 30. (You can change the maximum TTL from 30 to some other value with the –m command–line option.) Each group of three packets is intended to trigger an error message somewhere along the path. The three packets with a TTL of 1 trigger ICMP Time Exceeded errors at the first router; those with a TTL of 2 trigger the errors at the second router, and so on until the final destination is reached. traceroute captures the error messages that come back from the routers, extracts the router's address from the error messages, and prints out a list of router addresses as a trace.

To detect when the end of the trace is reached, traceroute uses an invalid port number. This causes the host at the remote end to return an ICMP Unreachable Port error. When traceroute receives this error message, it prints out the address of the remote system that sent the error, and terminates the trace. Putting all of these error messages together, traceroute produces a trace like the one shown in Listing 13.12.

Listing 13.12: Testing a Route with *traceroute*

```
$ traceroute turtle.big.edu
traceroute to turtle.big.edu (127.18.1.9), 30 hops max,
    38 byte packets
 1  172.16.55.254 (172.16.55.254) 1.424ms 3.187ms 1.295ms
 2  fgw225.chcc.org (172.16.2.232) 1.156ms 1.983ms 1.374ms
 3  igw225.chcc.org (172.16.5.254) 4.703ms 3.754ms 3.080ms
 4  Hssi2-1-0.GW1.TCO1.ALTER.NET (137.39.34.161)  9.511ms
    14.801ms  13.224ms
 5  115.ATM2-0.XR1.TCO1.ALTER.NET (146.188.160.34) 7.955ms
    8.788ms  11.569ms
 6  193.ATM3-0.XR1.DCA1.ALTER.NET (146.188.160.101) 7.794ms
    8.894ms  12.435ms
 7  195.ATM1-0-0.BR1.DCA1.ALTER.NET (146.188.160.225)
    13.934ms  13.076ms  14.229ms
 8  dca5-core1-s3-0-0.atlas.digex.net (209.116.159.97)
    87.888ms  100.783ms  96.771ms
 9  dca5-core3-pos1-1.atlas.digex.net (165.117.51.102)
    112.287ms  117.283ms 111.309ms
10  dca6-core1-pos4-3.atlas.digex.net (165.117.51.2)
    106.369ms  *  99.697ms
11  dca1-core10-pos1-2.atlas.digex.net (165.117.51.189)
    106.123ms  113.786ms 107.322ms
12  dca1-core5-pos5-0-0.atlas.digex.net (165.117.59.2)
    111.850ms  *  110.117ms
13  dca1-core2-fa6-0-0.atlas.digex.net (165.117.16.2)
    115.335ms  99.689ms  87.126ms
14  209.49.104.194 (209.49.104.194)  110.538ms  116.827ms
    106.463ms
15  1.atm1-0-0.csc0gw.net.umd.edu (128.8.0.223)  105.528ms
    105.439ms  107.240ms
16  turtle.big.edu (127.18.1.9)  118.430ms  105.404ms
    101.596 ms
```

This example shows a trace from our imaginary network to a system at a university. Each line indicates a hop along the path to the destination. The round−trip travel time of each packet is also printed. Note that there are three packets sent to each hop along the path. If a packet is lost (that is, if no error is returned for the packet), an asterisk (*) is printed instead of a round−trip time. If a series of three asterisks is printed on several lines, it indicates that the trace was unable to reach the remote end. The last router that responds is probably the last router that can be reached along the path.

Tip One row of asterisks is not enough to tell you there is a problem because some routers don't return ICMP errors. I let four or five lines of asterisks print out before I believe there is a remote routing problem, and then only if ping also fails.

In theory, every router that handles a traceroute packet responds with an error, and an accurate trace of the route is produced. Reality is a little different. Some routers silently discard the packets and return no error. Sometimes, different packets take different routes. If you examine the output of a traceroute command too closely or take it too much to heart, you'll be making a mistake. Use traceroute as a guide to where potential problems exist, but don't assume that it is completely accurate. See if the trace reached the remote site; if it didn't, see where it stopped. Those are the most meaningful things you can get from a traceroute.

Tip Routing is a two−way street. If possible, have the administrator of the remote system test the route from the remote server back to your system.

Analyzing Network Protocols

Linux provides some test tools for checking the state of TCP/IP protocol connections or examining the protocol interactions as they take place on the wire. System administrators often wonder why they would want to do this. The basic TCP/IP protocols have been in use for 20 years and don't require any debugging. Even if your system uses some new protocol that does have a bug, it is unlikely that as a system administrator you have the time or inclination to try to debug a protocol.

These things are true, but these tools have uses beyond debugging a protocol. The most important role for protocol analysis is as a tool to gain more information and insight about a network problem. Discovering that a connection terminates during the parameter negotiation can steer you toward checking the configuration, or discovering that a connection hangs without a clean termination can point you to a malfunctioning wrap−up script. Protocol analysis may be the last tool you use, but there are times when it is very helpful.

Checking Socket Status with *netstat*

netstat is a command that can be used to check on a wide variety of network information, such as the status of network connections, the contents of the routing table, what masqueraded connections are supported by the system, and what multicast groups the system has joined. The most important of these is the status of network connections, which is the default netstat display. To limit that display to TCP/IP network connections, use the −−inet command−line option, as shown in Listing 13.13.

Listing 13.13: Displaying Network Socket Connections

```
$ netstat  --inet
Active Internet connections (w/o servers)
```

```
Proto Recv-Q Send-Q Local Address Foreign Address   State
tcp        1      0 robin:1967    www.sybex.com:80  CLOSE_WAIT
tcp        1      0 robin:1966    www.sybex.com:80  CLOSE_WAIT
tcp        1      0 robin:1964    www.sybex.com:80  CLOSE_WAIT
tcp        1      0 robin:1963    www.sybex.com:80  CLOSE_WAIT
tcp        0    126 robin:23      phoebe:1449       ESTABLISHED
```

This command lists the currently active IP connections. Each line displays the transport protocol being used, the number of packets in the send and receive queues, the local address including port number, the remote address including port number, and the status of the connection. In Listing 13.13, the first four lines describe outbound connections to well–known port number 80. From reading Chapter 6, "The Apache Web Server," you know that 80 is the web server port. So those are outbound web connections. The last line in the sample shows an inbound connection to port 23: the telnet port.

The State field on each line indicates the TCP protocol state for that connection. Table 13.1 lists the possible TCP protocol states that netstat displays.

Table 13.1: TCP Protocol States

State	Meaning
CLOSED	The socket is completely closed.
CLOSE_WAIT	The remote end is shut down, but the local socket is not yet closed.
CLOSING	Both ends of the connection are shut down, but the local system still has data to send.
ESTABLISHED	The connection is established.
FIN_WAIT1	The local end of the connection is shutting down.
FIN_WAIT2	The socket is waiting for the remote end of the connection to shut down.
LAST_ACK	The protocol is waiting for the final acknowledgment on a closed socket.
LISTEN	The socket is listening for incoming connections.
SYN_RECV	A connection request has been received.
SYN_SENT	A connection attempt is underway.
TIME_WAIT	The socket is closed, but is waiting to clear remaining packets from the network.
UNKNOWN	netstat cannot determine the state of the socket.

In Listing 13.13, the inbound telnet connection has a state of ESTABLISHED, meaning that it is a healthy, active connection. The three outbound connections are all sitting in CLOSE_WAIT. All of these connections are directed at the same remote web server. The probable cause for this is a user with a browser open to the remote server that is not actively requesting data. Perhaps the user is reading the data; perhaps the user is out to lunch. In either case, the user has left the browser running. This is normal and causes no harm, other than consuming a port number. When the user closes the browser, these ports will close.

With the –a option, netstat displays all sockets (not just those that are active), and it does not have to limit the display to IP sockets. Listing 13.14 is an excerpt of the full socket listing from a Linux system. This listing is only about half of the number of lines actually displayed. To see the full listing, enter the netstat command on your own Linux system.

Listing 13.14: Display All Sockets

```
# netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address  Foreign Address State
tcp        0      2 parrot:telnet  robin:1027      ESTABLISHED
tcp        0      0 *:netbios-ssn         *:*      LISTEN
tcp        0      0 *:www                 *:*      LISTEN
tcp        0      0 *:smtp                *:*      LISTEN
tcp        0      0 *:1024                *:*      LISTEN
tcp        0      0 *:printer             *:*      LISTEN
tcp        0      0 *:imap2               *:*      LISTEN
tcp        0      0 *:login               *:*      LISTEN
tcp        0      0 *:shell               *:*      LISTEN
tcp        0      0 *:telnet              *:*      LISTEN
tcp        0      0 *:ftp                 *:*      LISTEN
udp        0      0 parrot:netbios-dgm    *:*
udp        0      0 parrot:netbios-ns     *:*
udp        0      0 *:netbios-dgm         *:*
udp        0      0 *:netbios-ns          *:*
udp        0      0 *:1024                *:*
udp        0      0 *:talk                *:*
raw        0      0 *:icmp                *:*           7
raw        0      0 *:tcp                 *:*           7
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags   Type    State     I-Node Path
unix  1      [ ]     STREAM  CONNECTED 415    @00000019
unix  1      [ ]     STREAM  CONNECTED 888    @0000003e
unix  0      [ ACC ] STREAM  LISTENING 519    /dev/printer
unix  0      [ ACC ] STREAM  LISTENING 725    /dev/gpmctl
unix  0      [ ACC ] STREAM  LISTENING 395    /dev/log
unix  1      [ ]     STREAM  CONNECTED 889    /dev/log
```

The first line in this listing shows an active inbound telnet connection, just like the one seen earlier. The next several lines all have the status LISTEN. These are the TCP services that this system offers. If the list of services produced by netstat on your server does not match the services that you think your system offers, you need to check the server's configuration. The asterisks in the address fields mean that any address is accepted.

Next come the UDP services offered by the system. UDP is a connectionless protocol, so it does not maintain connection state. For all UDP entries, the State field is empty. Again, these services should match the services you think you're offering.

For network testing, you can ignore the rest of the listing. It contains two entries for *raw sockets*, which are sockets that communicate directly to IP without using a transport protocol, and several entries for Unix sockets. The Unix sockets define sockets–based I/O for Linux devices and are not related to the TCP/IP network.

Use netstat to check the socket status when inbound or outbound connections appear to hang. An example of how this netstat information can be used to diagnose a problem occurred when we noticed strange symptoms on my campus e–mail server. The CPU utilization was very high. The mail queue was taking forever to process, and several users were having trouble downloading their mail. Nothing really appeared to be wrong with the network until the netstat command showed hundreds of connections in SYN_RECV state—the classic symptom of a SYN flooding denial of service attack!

All of these connection attempts were originating from the same system on our internal network. The router was reconfigured to block connections from that specific system, and immediately the

mail server began to recover. The administrator of the offending system was called and told that an intruder might have broken into his computer. That, however, was not the case. The offending computer was an experimental, massively parallel computer with hundreds of processors. A mistake in the experimental software caused the system to simultaneously start hundreds of connection requests whenever the system tried to connect to a remote host. The experimenters removed the test system from the live network, and we lived happily ever after. Using netstat to discover the problem and route filtering to block the offender, the initial problem was solved in just a few minutes.

The Linux version of netstat has an interesting option missing in many other netstat implementations: the –p option. When netstat is run with the –p option by the root user, it displays the PID and program name of whatever is using each socket. This can be very useful, particularly when you suspect a problem.

Watching the Protocols with *tcpdump*

tcpdump reads every packet from the Ethernet, and compares it to a filter you define. If it matches the filter, the packet header is displayed on your terminal, which permits you to monitor traffic in real time. Listing 13.15 provides a simple tcpdump example.

Listing 13.15: A *telnet* Handshake as Seen by *tcpdump*

```
# tcpdump host 172.16.5.1 and 172.16.24.1
tcpdump: listening on eth0
10:46:11.576386 phoebe.1027 > wren.telnet: S
   400405049:400405049(0) win 32120
   <mss 1460> (DF)
10:46:11.578991 wren.telnet > phoebe.1027: S
   1252511948:1252511948(0) ack 400405050 win 32120
   <mss 1460> (DF)
10:46:11.773727 phoebe.1027 > wren.telnet: .
   ack 1 win 32120 <nop> (DF)
```

This example shows a successful TCP three–way handshake between wren and phoebe. TCP is a connection–oriented protocol. Before TCP data can be sent, the connection must be established with a three–way handshake. First, the system requesting the connection sends a *synchronize sequence numbers* (SYN) packet to the destination host. The packet contains the sequence numbers that will be used by the source as well as other parameters such as the transmission window size (win) and the maximum segment length (mss). If the destination system will accept the connection, it sends a SYN (S) acknowledgment (ack) packet that includes the sequence numbers the destination will be using. Finally, the source sends a packet acknowledging the packet received from the destination, and the connection is under way. The packets exchanged on your network may contain many more optional packets, but the handshake will always be a SYN (S), SYN (S) acknow–ledgement (ack), and acknowledgement (. ack).

Each TCP packet displayed by tcpdump begins with a time stamp, followed by the source and destination address. From the first packet sent in Listing 13.15, you can tell that phoebe is the source of the connection, and is attempting to connect to the telnet port of wren.

Next, the flag field of the TCP header is displayed. In the example, the first two packets have a flag value of S, which means the SYN bit is set. That setting indicates that this is a connection request and that the computers are synchronizing sequence numbers. (The first packet is the SYN packet, and the second packet is the SYN ACK packet.) The next fields in the first two packets are the sequence numbers being used. (400405049 in the case of phoebe and 1252511948 in the case of

wren.) The example also indicates that phoebe has requested a windows size of 32120 bytes, a maximum segment size of 1460 bytes, and that its packets not be fragmented (DF).

Note Clearly, the details of this display don't make much sense unless you have a detailed understanding of the TCP/IP protocols. To further complicate matters, other protocols will produce other display formats, because they have different header formats. If you really want to tackle the details of such a display, you should read *Internetworking with TCP/IP: Principles, Protocols and Architecture, Vol. 1*, by Douglas Comer (Prentice–Hall, 2000).

The example shows only the first three packet headers. Immediately after the handshake, many packets are exchanged. If the example really stopped after the three–way handshake, the connection would be a failure. A failure at this point in the connection indicates that the remote system does not offer the requested service. Perhaps the service is not installed or is blocked for security reasons. Regardless, a failure at this point clearly shows that the remote system does not allow connections to the requested port.

A test designed to debug a real protocol problem might involve hundreds or even thousands of packets. Attempting to debug a complex protocol problem requires a great deal of technical skill, which most system administrators have; and a great deal of time, which no system administrator has. Ignore the details, and look for gross failures that might indicate where the network problem occurs.

tcpdump Filters

In the tcpdump example, the filter is host 172.16.5.1 and 172.16.24.1, which captures all traffic going to or coming from these two IP addresses. A wide variety of tcpdump filters can be defined. Table 13.2 lists some basic IP filters that are available for tcpdump.

Table 13.2: tcpdump Packet Filters

Filter	Captures Packets
dst host | net | port *value*	Destined for the specified host, network, or port.
src host | net | port *value*	From the specified host, network, or port.
host *host*	To or from the specified host.
net address [/*len* | *mask* mask*]*	To or from the specified network. An optional address mask can be defined either as a bit length or a dotted decimal mask.
port *port*	To or from the specified port.
ip proto *protocol*	Of the specified protocol type. Valid protocols are icmp, igrp, udp, nd, or tcp.
ip broadcast | multicast	Either IP broadcast packets or IP multicast packets.

With these filters, you should be able to capture any packets you need to debug a network problem. But my advice is to keep it simple. Designing complex filters and analyzing large packet dumps can take more time than other simpler methods of attacking a problem.

The utility of Ethernet monitoring tools such as tcpdump is limited by the network architecture. Many Ethernet hubs and most Ethernet switches only send traffic out of a port that is bound for the single system attached to that port. A system attached to a standard port on a switch cannot monitor the traffic flowing between other systems with tcpdump because the traffic is not on the cable attached

to the system. All the system can see is its own traffic and broadcast traffic, as shown in this example:

```
[root]# tcpdump host 172.16.8.8 and 172.16.8.2
tcpdump: listening on eth0
09:37:25.541133 arp who-has duck tell robin

1 packets received by filter
0 packets dropped by kernel
```

In this example, the monitor sees the ARP request because it is a broadcast, but it does not see the ping traffic because that traffic is unicast. The system running tcpdump must be attached to a supervisory port that is allowed to view all traffic. On some switches, this is a physical port; on others, the supervisory port can be configured in the switch software. Listing 13.16 shows the same tcpdump filter, run on the same Linux system, after the network has been reconfigured to allow the Linux system to monitor the network.

Listing 13.16: Monitoring Traffic with *tcpdump*

```
[root]# tcpdump host 172.16.8.8 and 172.16.8.2
tcpdump: listening on eth0
10:04:50.341133 arp who-has duck tell robin
10:04:50.341133 arp reply duck is-at 0:e0:98:a:13:e1
10:04:50.341133 arp who-has robin tell duck
10:04:50.341133 arp reply robin is-at 0:50:ba:3f:c2:5e
10:04:50.341133 robin > duck: icmp: echo request
10:04:50.341133 duck > robin: icmp: echo reply
10:04:51.361133 robin > duck: icmp: echo request
10:04:51.361133 duck > robin: icmp: echo reply
10:04:52.371133 robin > duck: icmp: echo request
10:04:52.371133 duck > robin: icmp: echo reply
10:04:53.371133 robin > duck: icmp: echo request
10:04:53.371133 duck > robin: icmp: echo reply

12 packets received by filter
0 packets dropped by kernel
```

In Listing 13.16, the monitoring system sees the ARP broadcasts and the unicast ARP replies, as well as all of the ICMP traffic caused by the ping. Notice the time difference of the time stamps on the packets from the failed example to the example shown in Listing 13.16. Almost 30 minutes elapsed while the switch was reconfigured to allow monitoring. During a real emergency, 30 minutes is an eternity. Many network administrators preconfigure a port to act as a supervisory port. When a problem occurs, the administrator can plug the laptop that he uses for testing into the waiting port.

Testing Services

At the top of the protocol stack are the applications and services that the user needs. Most errors are reported when a user attempts to use a service and the attempt fails. To be complete, testing needs to include the services.

Usually, a service can be tested directly. To test a web server, connect to the server with your browser. To test an FTP server, use the ftp command to connect to that server. These are user–oriented services, so they come with user–oriented commands that you can use for your testing.

Some services, however, are intended to provide service to a remote computer instead of a remote user. Directly testing these services is slightly more difficult, but it is often possible by using telnet to connect directly to the server port. You have already seen multiple examples of this in earlier chapters. Refer to the examples of testing imapd and the POP daemon in Chapter 11, "More Mail Services."

If ping tests succeed, but tests involving a specific service fail, the problem is in the configuration of the service at either the client or server end. Client configuration is usually simple and easy to check, but the server configuration can be very complex. Some complex services, such as sendmail and DNS, include their own test programs. Chapter 5, "Configuring a Mail Server," provides detailed examples of using sendmail –bt to test the local sendmail configuration. The next section of this chapter looks at the tools that are available to test a DNS configuration.

Testing DNS with *nslookup*

nslookup is a test tool that comes with the BIND software. It is an interactive program that allows you to query a DNS server for any type of resource record and to directly view the server's response. This tool is useful for checking your own servers, but (even more important) it can be used to directly query remote servers. Notice the emphasis on the word *directly*. Using nslookup, it is possible to directly connect to a remote server to see how that server responds to queries without going through your local name server. This is important because it eliminates the possibility that errors in your local server's configuration are affecting the results. Any test program that allows you to separate local problems from remote problems is worth its weight in gold.

To illustrate what this means, Listing 13.17 contains an example of how to use nslookup:

Listing 13.17: Testing DNS with *nslookup*

```
% nslookup
Default Server:  wren.foobirds.org
Address:  172.16.5.1

> set type=NS
> example.org.
Server:  wren.foobirds.org
Address:  172.16.5.1

example.org        nameserver = goat.example.org
example.org        nameserver = shark.fish.org
example.org        nameserver = whale.example.org
goat.example.org   inet address = 172.32.3.2
shark.fish.org     inet address = 172.30.8.2
whale.example.org  inet address = 172.32.3.1
```

Begin by entering the nslookup command with no arguments. This starts nslookup in interactive mode, which is the best way to use it to debug a server problem. When nslookup starts, it is using your local server, as indicated by the Default Server message. You need the name server (NS) records to locate the name servers for the remote domain you wish to test, so set the query type to NS and then enter the domain name you want to query. In the example, the domain name is example.org, and the local name server returns all of the NS records for that domain, which identify three servers: goat and whale in the example.org domain and shark.fish.org.

Now that you know the authoritative servers, connect to one of them to run the next phase of the test:

Listing 13.18: Testing Continues

```
> server goat.example.org
Default Server:  goat.example.org
Address:  172.32.3.2

> set type=ANY
> dolphin.example.org
Server:  goat.example.org
Address:  172.32.3.2

dolphin.example.org   inet address = 172.32.3.8

> exit
```

To connect directly to the remote server, use the server command. In the example, we chose to connect to goat. Then, set the query to the type of resource records you're interested in. This can be the keyword ANY for all available resource records or any of the standard resource record types: address records (A), mail exchange records (MX), start of authority records (SOA), and so on. (See Table 4.2 in Chapter 4, "Linux Name Services," for the possible DNS record types.) The ANY query is particularly useful because it provides all of the information available from the name server.

A successful test tells you that the remote server is responding and can resolve the desired hostname. If the test fails completely, the user may have the wrong hostname. If the test works but your local server is having trouble with the hostname, the problem could be in your local server or one of the other remote servers.

Sometimes, remote severs get out of synchronization, so querying all of the authoritative remote servers is worthwhile when you have intermittent problems resolving a hostname. For example, assume that you're having intermittent problems resolving the hostname dolphin.example.org. You could begin with the identical test shown in Listing 13.17, but instead of entering exit after testing the first remote server, you could switch to another server and rerun the test:

```
> server shark.fish.org
Default Server:  shark.fish.org
Address:  172.30.8.2
> dolphin.example.org
Server:  shark.fish.org
Address:  172.30.8.2

*** shark.fish.org can't find dolphin.example.org:
    Non-existent domain
```

In this case, the second authoritative server disagrees with the first. goat resolves the query for dolphin to an address, but shark can't. The most likely cause for this problem is that the servers have two different copies of the zone file. As in the following example, check the SOA records on each system to see if the serial numbers are different:

```
> set type=SOA
> example.org.
Server:  shark.fish.org
Address:  172.30.8.2

example.org        origin = goat.example.org
   mail addr = amanda.goat.example.org
   serial=10164, refresh=43200, retry=3600, expire=3600000,
```

```
   min=2592000
> server goat.example.org
Default Server:  goat.example.org
Address:  172.32.3.2

> example.org.
Server:  goat.example.org
Address:  172.32.3.2

example.org        origin = goat.example.org
   mail addr = amanda.goat.example.org
   serial=10164, refresh=43200, retry=3600, expire=3600000,
   min=2592000

> exit
```

In this example, the serial numbers are the same. This is bad news. If the serial numbers were different, the problem might be a temporary one that would be resolved as soon as the slave server updated the zone from the master server. The fact that the serial numbers are the same but the contents of the zone files are different is a major problem that must be addressed by the remote domain administrator. Luckily, the SOA record tells you who that is. Send mail to amanda@goat.example.org, and report the problem. She needs to get this fixed!

The developers of DNS plan to drop nslookup at some point in the future. This is too bad because nslookup is a good tool. The interactive nature of nslookup allows you to work through a problem, adjusting the next test based on the results of previous tests. The disadvantage of nslookup is that it is difficult to script for repetitive tests. However, Linux provides two more DNS test tools that will continue to be maintained in the future and are suitable for scripting: host and dig.

Testing DNS with *host*

The host command is a very simple tool for looking up an IP address. The format of the host command is:

```
host [options] domain-name [server]
```

Only the host command and the domain name of the remote host are needed to look up an IP address. To look up different resource record types, specify the desired record type (mx, soa, ns, etc.) with the –t argument in the *options* field. To pass the query to a specific server, identify the server in the *server* field. If no server is specified, the local server is used.

Listing 13.19 is an example of the host command in action.

Listing 13.19: Testing DNS with the *host* Command

```
$ host -t any dolphin.example.org goat.example.org
Using domain server 172.32.3.2:
dolphin.example.org has address 172.32.3.8
$ host -t any dolphin.example.org shark.fish.org
Using domain server 172.30.8.2:
Host not found.
```

This is the same test that was run previously using nslookup. Two different servers, goat and shark, are queried for any type of DNS records relating to dolphin. goat replies with an address record, and shark replies with an error.

The host command is simple to use, to script, and to understand. As Listing 13.19 shows, it can be used for the same tasks as nslookup. In Listing 13.19, you discovered that the two servers give different answers. Therefore, your next step is to query each server for the zone's SOA record by rerunning the host command with new command–line options. Whether you use nslookup, host, or dig (which is discussed next) is primarily a matter of personal preference.

Testing DNS with *dig*

dig is another DNS test command that is very similar to host. It has the same strengths and weaknesses, but is both more powerful and more complex. The basic format of a dig command is

```
dig [@server] domain-name [type]
```

If a server is defined, the name of the server must be preceded by an @. If a server is not specified on the command line, the local server is used. The type of resource record being requested is identified using a standard record type or the keyword any; if a resource record type is not specified, the dig command fetches address records. For example, to query the server goat for any records pertaining to dolphin, enter

```
$ dig @goat.example.org dolphin.example.org any
```

A nice dig feature is its capability to make reverse domain queries simple. Remember that when IP addresses are mapped back to domain names, they are first reversed to make the structure compatible with domain names, and the domain name in–addr.arpa is appended to the end of the reversed address. To do a reverse lookup with nslookup, you first set the query type to PTR and then manually enter the reversed and expanded address. With dig, you just use the –x option, as shown in Listing 13.20.

Listing 13.20: Testing DNS with *dig*

```
$ dig -x 172.16.55.105
; <<>> DiG 8.2 <<>> -x
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 5
;; QUERY SECTION:
;;      105.55.16.172.in-addr.arpa, type = ANY, class = IN
;; ANSWER SECTION:
105.55.16.172.in-addr.arpa.  8H IN PTR  rail.foobirds.org.
;; AUTHORITY SECTION:
55.16.172.in-addr.arpa.  8H IN NS        dove.foobirds.org.
55.16.172.in-addr.arpa.  8H IN NS        hawk.foobirds.org.
;; ADDITIONAL SECTION:
dove.foobirds.org.          19h7m19s IN A   172.16.2.2
hawk.foobirds.org.          16m17s IN A     172.16.16.1
;; Total query time: 2 msec
;; FROM: rail.foobirds.org to SERVER: default -- 172.16.5.1
;; WHEN: Tue Jun 29 16:07:30 1999
;; MSG SIZE  sent: 43  rcvd: 213
```

This example shows something else about dig—it is very talkative. It displays everything that is exchanged between the DNS client and the server. The meat of the response is the answer section buried in the middle of the display, which says that the address 172.16.55.105 is assigned to rail.foobirds.org. The other parts of the display are the following:

- The query section, which displays the query sent to the server
- The authority section, which gives a list of the authoritative name servers for the domain that was queried
- The additional section, which provides the addresses of the authoritative servers

The sections in the dig output exactly match the sections in a DNS reply packet. This allows you to look inside the protocol exchange and see all of the information that the system receives in response to a DNS query.

dig, host, and nslookup are a powerful trio of tools for testing and debugging domain name service. When the error message is Unknown Host, a Linux system is well–equipped to tackle the problem.

In Sum

This final part of *Linux Network Servers* examined some of the ongoing tasks that are necessary for maintaining a reliable operational server. This book has focused on building a Linux network server from the ground up. Starting from configuring the network interface, you have seen how to lay the foundation for a reliable operational server. This book has described how the basic Internet services of routing, name service, e–mail service, and web service are configured, maintained, and secured. Further, you learned that a Linux system provides configuration services and file and printer sharing that are compatible with all types of clients, making Linux the operating system that can integrate your departmental network and thus simplify maintenance.

When configured and maintained by a well–informed computer professional, Linux is an excellent platform for an Internet or a departmental server. You provide the professional expertise, and this book has provided the information to get you started. As you dig deeper into specific services and tasks, you'll find the detailed information you need on many of these topics in the other books in the Craig Hunt Linux Library.

Appendices

Appendix List

Appendix A: Installing Linux
Appendix B: BIND Reference
Appendix C: The m4 Macros for sendmail

Appendix A: Installing Linux

Overview

Installing Linux, which is an essential part of getting a Linux network server operational, is relegated to an appendix because most readers of the Craig Hunt Linux Library have installation experience and have a system that is already running Linux. Installation is described in an appendix, not because it is unimportant, but because many readers do not need this information. Most of the books in the library are read by experienced Linux administrators. However, this book, like *Linux System Administration*, is one that Unix and Windows administrators will pick up first when they want to learn about Linux. Experience with Unix system administration does not give you experience installing Linux. If you have never installed Linux before, the appendix is for you.

In some ways, the initial installation of the Linux software distribution is more challenging than configuring the various network services after the operating system is installed. This appendix examines the basic installation tasks, and looks at the pitfalls that can make this the most frustrating part of building a Linux network server.

This appendix also illustrates one of the subtle reasons why Linux has a very low total cost of ownership. Some operating systems require you to master the installation of the operating system and then to master equally complicated installations to add remote access, routing support, or web service. By the end of this appendix, everything will be installed—no additional complicated installations are required—and you'll be ready to configure any service.

Administrators coming from Unix or Microsoft backgrounds often find Linux installation challenging because they have limited experience with installations that require hardware and software integration. Most organizations buy PCs with Microsoft Windows preinstalled and Unix servers with Unix preinstalled. This means that the hardware has been "prequalified" for the software by the vendor. Therefore, even if the administrator needs to reinstall or upgrade the operating system, they do not need to perform any hardware/software integration.

It is possible to buy computers with Linux preinstalled. This is often very cost−effective because the cost of in−house staff time is frequently higher than the small premium involved in purchasing systems with preinstalled software. Despite the cost−effectiveness of this approach, I don't recommend it for all installations. You should install Linux several times yourself, even if you use some systems with preinstalled software. The reason is simple: In order to properly maintain a Linux network server, you need to feel confident about installing and reinstalling the operating system.

To gain confidence, get an obsolete or underutilized system, and use it as a training tool. Take the system apart. Remove unneeded hardware. Add in the type of Ethernet card you'll use on your real server. Then install Linux repeatedly until you feel at ease about tearing apart hardware and installing Linux software.

In this appendix, Red Hat Linux 7.2 is used as the primary example, but the installation steps described here are taken for every Linux distribution. Regardless of the distribution you are working with, when you install Linux from a CD−ROM, the installation process is basically the same. You begin by planning the installation and creating any necessary boot materials. For some systems, this means creating boot floppies. You then reboot the system so that it is running Linux and then run the Linux installation program. Most Linux distributions automatically start the install program; on a few, you manually start the program. Either way, the system is booted so that a small Linux system is running before the real Linux installation begins. You then partition the disk, and load the

Linux software into the new disk partitions. When the loading is finished, you have a permanent Linux installation. The following sections discuss each of these steps in detail for Red Hat 7.2. The details vary for each distribution, but the overall pattern is the same.

Linux allows you to direct the installation and configuration process, and the installation program asks you to make many decisions. Most of these decisions are easy, but it is always best to be prepared ahead of time. We begin by planning for the installation.

Installation Planning

Linux is about choices—more choices than most other operating systems. Many Unix systems limit your choice of hardware vendors because the operating system runs on only one type of hardware. Microsoft Windows limits your choice of operating systems vendors because Windows is available only from Microsoft. With Linux, you can choose from many different distributions, all of which run on a wide array of hardware. Choices mean decisions, and thoughtful decisions require careful planning. Before you start an installation, collect all of the information needed to install and configure your system.

Hardware Information

Linux runs on almost any hardware, from a PDA to an IBM mainframe. If your organization is very large, and you plan to run Linux on a mainframe, your IBM salesperson will be happy to help you select and integrate your hardware. On the other hand, if you're like most of us, you will run Linux on PC hardware. If you assemble your own server from PC components, a few rules of thumb will help you avoid hardware problems during the installation:

- Check the "Linux Hardware Compatibility HOWTO." Also, check the home page of the Linux distribution you plan to use. See if the vendor lists any hardware incompatibilities before you buy. When possible, stick to the hardware list provided by the vendor.
- Don't purchase the very latest model of any piece of hardware. Most adapter cards are delivered initially with Microsoft Windows drivers. It takes awhile before drivers are available for other operating systems, including Linux. Avoid newly released hardware unless you know there is a fully functional Linux driver.
- A network server does not need high–performance graphics. You'll use the server console to run administrative applications, not video games. Don't invest in the latest high–performance 3D graphics or the largest high–resolution monitor. Purchase a well–established video card and monitor that are documented to work with Linux. Use the money you save for more memory or a faster CPU.
- Don't buy things you don't need. For example, network servers don't require sound cards or speakers. Avoid buying equipment that complicates configuration and adds no value.
- Never buy any "Win" hardware. Some internal modems are called "Winmodems" because they are designed to work with Microsoft Windows. Some printers, called GDI printers, depend on the Windows Printing System. These devices are so dependent on the Windows operating system that they cannot work without it.
- Sometimes, the best deals on PCs are on systems that come preconfigured with lots of features. If you find yourself with such a system, remove all of the unneeded equipment, and use it to enhance your desktop system. A cool sound system is a nice addition to your desktop, but it is unnecessary on a server. Superfluous equipment is just a source of trouble on a server system.

If you select your own hardware, you will know all you need to know about the hardware. Many times, however, someone else asks you to configure a system for them that you know very little about. Look at the documentation that comes with the system and, if Windows is installed, use the information from the Microsoft Windows configuration to learn about the hardware. If you have problems installing Linux with the information you gleaned from Windows and the documentation, don't be afraid to pull off the system case and examine the cards and devices installed in the system.

Linux properly detects most hardware. Generally, you can let Linux configure the hardware for you. However, there are times when some piece of hardware is not detected. Here is a list of the hardware information that is useful to have on hand if you encounter a hardware detection problem:

Hard drive characteristics Know the make, model, and capacity of each drive; and the drive geometry, which is the number of cylinders, heads, and sectors. Know whether it is a SCSI or IDE drive. If you have an IDE drive, know if it runs in Logical Block Addressing (LBA) mode, which maps the physical sectors on the disk drive to logical sectors. The documentation that comes with the disk drive should provide all of this information. If yours doesn't, check the settings in the BIOS setup program. If all else fails, you can always remove the drive from the PC to see if the drive characteristics are printed on the case— they frequently are.

SCSI adapter information Know the exact make and model of the adapter.

Ethernet adapter information Know the make and model of the Ethernet adapter. PCI adapters require no additional information. If it is an ISA adapter, however, you should also know the Interrupt Request (IRQ) number, the I/O port address, the adapter memory address, and (if used) the Direct Memory Address Request (DRQ) number. The documentation of an ISA adapter will show the default settings; Windows can tell you the current settings.

Video monitor characteristics Know the make and model of the monitor, and its technical specifications, including the monitor's horizontal and vertical sync ranges, as well as its maximum resolution. The monitor's documentation will provide the technical specifications.

Video interface characteristics Know the make and model of the video card, and the amount of video memory on the card. Additional information that may prove helpful is the make and model of the video chip set used on the card, and whether the card has a clock chip (and if it does, the model of the clock chip). (Separate clock chips are rarely used anymore.)

CD−ROM characteristics Know the type of interface: SCSI, IDE, or "other." If an "other" interface, such as a sound card, is used, the make and model of the CD−ROM is also needed. (Again, "other" CD−ROMs are extremely rare these days on any system, and should never be used on a server.)

Network Information

In addition to information about the system's hardware, you need information about the network in order to configure a networked system. If DHCP is used, the Linux system will automatically obtain the correct configuration from the DHCP server. If it is not used, manually provide the correct configuration during the installation. The required network configuration information includes

- The Internet address assigned to this system by the network administrator.
- The network mask used on your network.
- The hostname assigned to this system by the domain administrator.
- The domain name assigned to your organization.
- The addresses of the domain name servers.
- The default gateway address, unless a routing protocol is used. If a routing protocol is used, it is configured as described in Chapter 7, "Network Gateway Services," after Linux is installed.

Software Considerations

When planning a server installation, consider how the server will be used. Many servers are general−purpose systems that provide a wide range of services. Those systems might need all of the system software. Other servers, however, have a dedicated purpose that requires only some of the system software. Don't install everything on these dedicated systems. Choose only the software that is needed.

Linux systems group related software packages together to simplify the selection of the appropriate software. Red Hat provides several different software groups that it calls *software components*.

Each Red Hat software component has a descriptive name that helps you determine whether or not it is useful for your server. Several of the components are clearly identifiable as network server packages. For example, News Server, NFS Server, Anonymous FTP Server, Web Server, and DNS Server are all software components. From the name alone, it is easy to tell what package would be required for a dedicated DNS server.

Other components provide the software packages needed for the client side of a network connection. Mail/WWW/News Tools, DOS/Windows Connectivity, Networked Workstation, Dialup Workstation, SMB (Samba) Connectivity, and IPX/NetWare™ Connectivity provide traditional Unix tools, as well as tools to connect to NetBIOS and NetWare servers.

Finally, there are many components, such as the software development components, that are not directly related to a network server but may be needed on your system for development and maintenance. In the planning phase, it is useful to think of all of the groups of software that will be required for your server to fulfill its purpose.

Selecting an Installation Method

Linux can be installed from several different sources: FTP (File Transfer Protocol), NFS (Network File System), SMB (Server Message Block), HTTP (Hypertext Transfer Protocol), local hard drive, or CD−ROM. A server system is often installed from a CD−ROM. It is a simple and fast installation method, and the CD−ROM provides a reliable backup medium when you need to reinstall.

Never install a server from a local hard drive. This is an obsolete installation method that involves copying the operating system twice: first to a DOS partition on the local disk and then, as part of the actual installation, from there to the Linux partition. This installation method dates from the time when most systems had neither a CD−ROM nor a network interface.

All of the other installation methods depend on a network interface. FTP is primarily used to download files from the Internet, and the FTP method installs via the network from an FTP server. This method is used to directly load free Linux software across the Internet to a desktop system. Of course, downloading from the Internet is too slow and unreliable for a production environment. If

you decide to use FTP installation, place the Linux files on an anonymous FTP server on your local area network, and use the FTP installation mode to install systems attached to that network.

The HTTP installation method is very similar to the FTP method, and has similar constraints. The only real difference is the protocol used to move the files across the network: One uses FTP and the other uses HTTP.

If you use a network installation method, it will probably be either NFS or SMB. Details on configuring NFS and SMB are found in Chapter 9, "File Sharing."

SMB is the protocol used by NetBIOS servers for file sharing among Microsoft Windows systems. If the primary file server on your network is a Windows NT/2000 server, you can place the Linux distribution files on that server, and use SMB to install Linux on your network clients.

Most likely, you will use NFS rather than SMB to share files between two Linux systems. NFS is the most popular protocol for sharing filesystems among Unix computers. Place the Linux distribution files on a Linux server, and your desktop Linux clients can use NFS to install Linux from that server. Most distributions can be placed on the server simply by copying files from the Linux CD−ROM to the directory on the server that is exported to the clients, which can be done with a cp command. However, if NFS is used to install Red Hat 7.2, copying the individual files is not enough: The image of the CDROM must be stored on the server. Images are copied using the dd command. In the section "Creating Additional Installation Disks," we use the dd command to copy a floppy disk image from storage onto a floppy disk. A similar dd command could be used to place an image of a CD−ROM onto a NFS server.

All of these network−based installation methods are most commonly used at sites that manage a large number of Linux systems. Stand−alone servers are most often installed from local media. By default, Red Hat uses the CD−ROM installation method, and it expects your system to boot from the installation CD−ROM. If your system cannot boot from a CD−ROM, the last step in planning for the installation is to prepare all necessary boot materials.

Making a Boot Disk

A computer that cannot boot from a CD−ROM needs a boot floppy. Not every Linux distribution ships with a boot disk, even when you buy the top−of−the−line boxed set. Often, when a boot disk is needed, you have to make your own.

Making a boot floppy is simple, and the process is essentially the same for all Linux distributions: A boot image is copied from the CD−ROM to the floppy using either rawrite under DOS or dd under Unix. An example of each command illustrates how they are used.

In the Red Hat 7.2 distribution, the boot image is stored in the images directory, and the rawrite program is stored in the dosutils directory. Listing A.1 creates a Red Hat boot disk from files stored on a CD−ROM mounted on DOS drive D:

Listing A.1: Using *rawrite*

```
D:\>dosutils\rawrite
Enter disk image source file name: images\boot.img
Enter target diskette drive: a:
Insert a formatted disk into drive A: and press -ENTER- :
D:\>
```

Creating Additional Installation Disks

Boot disks are not the only disks that might be needed for an installation. Although a server installation probably will not require additional installation disks, you may be called upon to install a system that does. A laptop that needs to install Linux through a PCMCIA network adapter or through a CD−ROM drive attached to a PCMCIA SCSI controller is an excellent example of a system that may need an additional installation disk. For example, to install Red Hat 7.2 on such a laptop, you need a pcmcia.img disk. The sample code in Listing A.2 creates that disk on a Linux system:

Listing A.2: Creating Floppy Disks with *dd*

```
# mount /dev/cdrom /mnt/cdrom
mount: block device /dev/cdrom is write-protected, mounting read-only
# cd /mnt/cdrom/images
# dd if=pcmcia.img of=/dev/floppy
2880+0 records in
2880+0 records out
```

The first line of this listing attaches a physical device to a mount point with the mount command. The device, /dev/cdrom, is the CD−ROM drive. The mount point, /mnt/cdrom, is an empty directory on the Linux system. The directory structure of the CD−ROM is now available through the mount point, as illustrated by the change directory (cd) command that puts us in the images directory of the CD−ROM. The actual creation of the diskette is done by the dd command, which copies the input file pcmcia.img to the output file /dev/floppy. In this case, the output file is a physical device. On many Linux distributions, the device name /dev/floppy is equivalent to the device name /dev/fd0 because /dev/floppy is often symbolically linked to /dev/fd0.

After you have selected the hardware, planned the software, and prepared the boot materials, you're ready to run the installation. The first step is to boot the installation program.

Booting the Installation Program

Insert the Linux CD−ROM in the CD drive, and turn on the computer. On most computers, that's all it takes to boot the Red Hat installation program.

When you boot the installation program, a screen full of information is displayed, and you are given a boot prompt. The information you receive varies from distribution to distribution. In our example, Red Hat uses it to tell you about the installation program and to provide a help feature.

But the important thing on this screen is not the information; it's the boot prompt. Most distributions have it, and it serves the same purpose for them all: It gives the system administrator an opportunity to provide input to the boot process. For example, to run a expert−style text mode installation on a Red Hat system, enter

```
boot: text expert
```

This bypasses the default graphical installation program, and starts the text−based installer. The expert option causes the installation program to present more configuration options. Most installations do not require either text mode or expert mode. This is just an example of how the boot

prompt is used, and the boot prompt can do much more than control which installation program is used. For example, it can be used to input the correct geometry for the hard disk.

Most systems don't need input at the boot prompt, so the first time you run the installation program on any system, just press the Enter key. Only if the installation fails should you try it again with input at the boot prompt. An example of this is if the installation program was not able to automatically detect all of the storage on the hard drive through normal probing. This can happen with very large disk drives on systems in which the BIOS reports the wrong disk geometry to the operating system. See Chapter 1, "The Boot Process," for more information about the commands that can be entered at the boot prompt.

Responding to the boot prompt boots the Linux kernel, which in turn starts the Linux installer. Linux will detect some essential hardware during this phase. As it does, it displays information about that hardware on the screen. At this time, Linux also constructs a runtime environment for the installation program. If you used a boot floppy, the installation may prompt for a second disk to create the runtime environment or to install optional device drivers.

When the system completes the initialization of the runtime environment, the Red Hat installation asks you to select the language and keyboard you're using, and then displays a welcome message. If it detects a PCMCIA chipset, it asks you if you need PCMCIA support. At this point in the installation process, PCMCIA support is only required for laptops that have a CD−ROM attached through a PCMCIA interface card or for laptops that are installing over the network using a PCMCIA Ethernet card. If you answer that you do need PCMCIA support, you must provide a supplemental disk. Create it ahead of time with rawrite or dd by copying /images/ pcmcia.img from the Red Hat CD−ROM.

If the system did not boot from the CD−ROM, the Red Hat installation next asks for the installation method. (To force a system that boots from the CD−ROM to ask for an installation method, enter **text expert** at the boot prompt.) If you select the Local CDROM installation method, make sure the Red Hat CD−ROM is in the drive, because the installation program attempts to detect the correct drive by locating the CD−ROM. If the installation program cannot detect the CD−ROM in an IDE drive, it will ask you what type of CD−ROM drive you have. The choices are SCSI and Other. Some servers use a SCSI CD−ROM drive. If you select SCSI, you will be asked to tell the installation program which SCSI device is the CD−ROM. The Other category is reserved for CD−ROM drives connected through sound cards or other proprietary interfaces. If you select Other, you will be given a list of supported CD−ROM drives. Select the make and model of your drive from that list.

Note If you have an IDE drive that was not detected, you must reboot the system and pass the name of the CD−ROM device to the installation program at the boot prompt (for example, hdc=cdrom).

After the CD−ROM is located, the Red Hat installation asks if this is an install or an upgrade. The first time you install Red Hat on the hardware, select Install. For subsequent installs, you can use the Upgrade Existing System option to save the configuration files you have customized for your system, although some administrators prefer to always do a complete installation and to restore customized configuration files from the system backup. When an upgrade installation is used, new configuration files are given the extension .rpmnew so that they do not overwrite your configuration; or, in some cases, the old file is given the extension .rpmsave, and the new one is put into place. For example, after an upgrade the new httpd.conf file is stored in httpd.conf.rpmnew.

When you select Install, you must also select the type of installation:

- Workstation installation is designed for a desktop system.
- Server installation provides a predefined server configuration.
- Laptop installation ensures that packages needed for a laptop are installed.
- Custom installation gives you the most control over the installation and configuration process.

As the system administrator, you should know as much as possible about how your server is installed and configured, so select Custom. Choosing Server will work, but you won't have as much control over the installation.

Partitioning the Disk

Red Hat offers two different disk–partitioning tools: Disk Druid and fdisk. Many Red Hat administrators use Disk Druid, but fdisk is used by the administrators of all Linux distributions. Though you'll use only one of these tools during the installation, this section will discuss both so that you're prepared to install any Linux distribution.

For most administrators, partitioning the hard drive is the installation task that creates the most tension. It doesn't need to be, particularly for a server installation. People worry about partitioning because they do not want to lose the data that is already on the disk.

The first time you install a server, there isn't anything on the disk that you want. Even if the PC came from the hardware supplier with Microsoft Windows preinstalled, it doesn't matter. You don't need it. Sometimes, desktop client systems can *dual–boot*, which means that they have more than one operating system installed, and the user of the system boots the different systems for different applications. Servers do not dual–boot. A server cannot be offline running Microsoft Windows when a client needs service. Therefore, extra operating systems installed by the hardware vendor are not needed. Because of this, you can start your server with clean disks, and you can partition your disks with less worry.

Working with a Windows Partition

When you install a server, you don't need to worry about retaining the data in a Windows partition. However, not every installation is a server installation. Frequently you *do* need to be concerned about the Windows partition.

To partition a disk that contains Windows 9*x*, follow these steps:

1. Save the Registry.
2. Run a full system backup of Windows.
3. Defragment the hard drive.
4. After the disk is defragmented, check how much disk space is currently used. Add to this figure the amount of growth you want to allow for Windows. This gives you the minimum size for the Windows partition.
5. Copy the fips program from the Linux CD–ROM to the c:\windows\temp directory. (I always copy fips to the hard drive because I find that many users do not configure CD–ROM support for DOS mode.)
6. Reboot the PC in DOS mode.
7. Run fips.

The fips program splits the hard disk into two partitions. The first, called the *old partition*, contains Windows, and is just large enough to hold the data currently stored by Windows. Use the cursor keys to adjust the size of the old partition until it is at least the minimum size you calculated for the Windows partition. The remaining space, which fips calls the *new partition*, is the space that is available for installing Linux.

fips is provided without warranty, and is not supported by any of the Linux vendors, so, though it has been used many times with great success, no one guarantees it will work for you.

fips runs under DOS, and works only on File Allocation Table (FAT) filesystems. Systems such as Windows 2000 and Windows XP, which cannot boot into DOS mode, cannot run fips. However, these systems can define partitions when they are installed. Additionally, there are commercial partitioning products that can be used with these operating systems. A product that I have used to partition a disk without deleting an existing Windows partition or reinstalling Windows is Partition Magic. It is a well–documented, easy to use commercial product, and there are other similar products as well. Regardless of what tool you use, always back up your data before using any partition tool.

Partition Planning

Before partitioning a disk drive, plan exactly how you want it partitioned. To do this properly, you need to understand what partitions are and why they are used. If you have a Unix background, this is probably something you understand well, but if you're coming from a Microsoft background, an explanation is probably needed.

Despite the fact that the DOS partitioning program and the Linux partitioning program share the same name (fdisk), the concept of partitioning for these two operating systems is subtly different. The DOS fdisk program divides a large disk drive into smaller logical drives. Using fdisk, for example, the C drive might become both C and D. Each of these logical drives has it own root filesystem, and looks to the operating system and to the user as if they are physically separate devices. Conceptually, the DOS fdisk command is used to divide things.

On the other hand, the Linux fdisk command is part of a system that unifies things. Linux (and all other Unix–like operating systems) unifies all of the physical disk drives under a single root. Instead of seeing a separate device or partition as a logical drive (C or D), the user sees the partition as a directory in the filesystem, such as /home.

The directory to which a device or partition is "attached" is called a *mount point.* This term springs from the fact that Unix has been around for a long time. Back in the dark ages, a computer operator had to manually mount a "disk pack" on the spindle in the disk drive assembly. Disk packs are no longer used, but the flexibility of mounting and dismounting physical devices to directories within the filesystem lives on in the *mount* and *umount* commands.

The partition table you create is written directly to the disk drive. The mount points and the partitions that are mapped to them are written to the /etc/fstab file from where they are read during the boot process, and mounted using a mount –a command. The mount command is covered in detail in Chapter 9.

The philosophy of the DOS and Linux filesystems may be different, but the ultimate purpose of partitioning is the same. Partitions keep incompatible data separated, and they divide a large

storage space into pieces that are more manageable and can be more effectively used. At a minimum, Linux requires two partitions: a swap partition and a root partition.

Most administrators use more than two partitions. Providing a partition to hold user files and another to hold the operating system software is an example of what administrators do to organize and manage their disk space. Table A.1 lists the most common partitions and describes the purpose of each.

Table A.1: Common Partitions

Partition Name	Description
swap	Required. It holds the swap space for the operating system.
root (/)	Required. The root is the foundation of the entire filesystem.
/boot	Some distributions place the boot files in a separate directory so that a boot partition can be created separate from the root partition.
/usr	The usr partition contains most of the system software.
/home	The home partition contains all of the user home directories and almost all of the user files.
/var	The var directory holds all the printer spool files, the mail files, the news file, and the system log files.
/opt	The opt directory holds optional software. Some third–party software packages assume that the /opt directory is available, and install themselves there by default.

Swap Partitions

The *swap partition* is needed to provide swap space for the operating system. Linux, like all Unix operating systems, uses *virtual memory*. In other words, it uses disk storage as an extension of the RAM memory. Inactive processes are swapped out of memory and held on disk whenever the RAM they occupy is needed by an active process.

The disk drive, and therefore swapping, is very slow. You want to design your server to avoid swapping to the greatest extent possible. If you need more memory, add more RAM—not more swap space. Swap space should be needed only for spikes of activity that temporarily create an unusually large number of processes.

The swap space should be twice the size of real memory; however, the less real memory available to your machine, the more important swap space becomes. For systems with 256MB or less memory, you should always plan for a swap partition twice as large as memory. If the memory is greater than 256MB, a swap space that is the same size as the memory might be adequate for your server.

Additionally, if you have disks on multiple controllers, you can speed up swap access by having a small swap partition on each disk rather than just one big one. For example, assume that you want 512MB of swap space and you have four disks; you could create a swap partition of 128MB on each disk. Linux will automatically round–robin between them all, resulting in faster access times.

Root Partitions

Planning the swap partition is easy: You should have one, and it will probably be twice the size of real memory. Planning the other partitions takes a little more thought. There are two basic schools

of thought on partitioning: one school says to take all of the remaining disk space and create one large root partition, and the other says to create several carefully–sized partitions. Each school of thought has some merit.

The main advantage of creating a single large root partition is that it is easy. It also avoids the "root file system full" error that occurs when /tmp or /opt grow too large. If the entire disk is dedicated to the root directory, you won't have a full root filesystem until the entire disk really is full of data.

The problem with this approach is that it is *too* easy. It doesn't take the variability and load of a server into account. It works fine for a single–user client system with a single disk drive, but a server environment is more complex:

- A server usually has more than one disk drive. A single partition cannot span multiple disk drives.
- A server supports many users. The /home directory holds the files for these users, and often grows very large.

A single partition is simply not flexible nor reliable enough for a server. Placing all files in the root partition reduces the reliability of that critical partition by increasing the chance that the partition may become corrupted.

The root partition is critical because it contains the files necessary to boot the system. Some Linux distributions consider this important enough to place the critical boot files in /boot so that a separate partition can be created just for them. Additionally, placing the critical boot files in a separate /boot partition creates a partition that is small enough to place almost anywhere on the disk. If either the Workstation or Server installation classes are selected on a Red Hat system, a /boot partition is created. The Workstation installation class creates a swap partition, a /boot partition, and a root partition. This is a slight variation on the "single large root" partition scheme. The server installation class creates a swap partition, a /boot partition, a root partition, a /usr partition, a /home partition, and a /var partition. Although different Linux vendors recommend different partitioning schemes, they all recommend more than one partition for a server.

A Basic Server Partitioning Scheme

It is possible to create too many partitions. This creates a confusing structure, and can waste space. The Red Hat server installation class provides a reasonable example of partitioning for a server. A basic partition structure for multiuser, multidisk server based on the Red Hat scheme is

- A swap partition that is twice the size of real memory.
- A root (/) partition of 500MB.
- A /var partition of 300MB. Create a /var/tmp directory, and symbolically link /tmp to it. The size of the /var partition can vary greatly, depending on the number of users who leave their mail on the server, and whether or not the server is a news server. Network news consumes an enormous amount of space—many gigabytes every day. If this is the enterprise news server, you need a larger /var partition. However, if the users don't store mail on the server and it isn't the news server, a separate /var partition is not needed at all. A 300MB /var partition is adequate for most departmental servers, particularly if the growth of log and spool files is closely monitored.
- A /usr partition of 3GB. Create a /usr/local directory to contain local software. Create a /usr/opt directory, and symbolically link /opt to it. 3GB is adequate for the Linux software, but if you plan to install lots of large third–party software packages, you may need a /usr partition that is twice as large.

- User data is placed in the /home directory. If you have multiple disk drives specifically to hold user data, create a /home/user1 partition encompassing the remainder of the first disk drive, a /home/user2 partition encompassing the entire second disk drive, and so on. Segmenting the directory in this way allows it to span multiple disks.

  Tip To simplify the pathnames for the users and to simplify moving users between physical disks, create a small /home partition on the first disk that contains only symbolic links to the users' real home directories that reside on other disks. User accounts are covered in Chapter 3, "Login Services."

Symbolic Links

In Linux, directories can be "linked" together by creating a pointer from one directory to another. These pointers are called *symbolic links*. Use the ln command to create symbolic links. For example, to create a "directory" named /opt that links to the real directory /usr/opt, enter

```
ln -s /usr/opt /opt
```

After this symbolic link is created, references to the link /opt have a corresponding effect on the directory /usr/opt. An ls of /opt lists the directory contents of /usr/opt. A file stored in /opt is really stored in /usr/opt. Symbolic links increase the flexibility of the Linux filesystem.

No suggestions from anyone, no matter how experienced, can take the place of your own judgment. You know what you want to do with your server, you know how many people it will serve, and you know the tasks it will perform. Use this knowledge to plan your partitions.

After you have developed a partition plan, you need to write the partition scheme onto the hard disk. Red Hat provides two tools to do this: the widely used Linux fdisk program and the Red Hat program Disk Druid.

Partitioning with Disk Druid

If you select a custom installation, the Red Hat installation program asks which partitioning tool it should use. Select either the traditional fdisk program or Disk Druid, which uses a full–screen interface (see Figure A.1). When run under the graphical installer, Disk Druid uses a basic point–and–click interface.

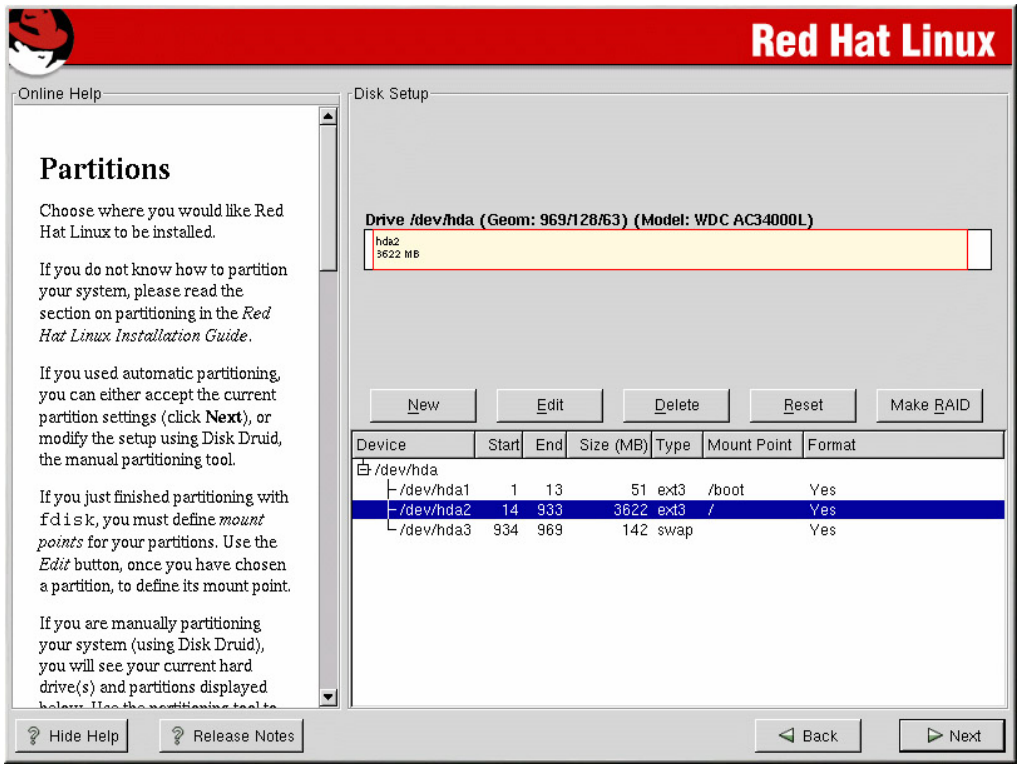


Figure A.1: Disk Druid's main screen

The top of the screen displays the device name, the disk geometry, and the make and model of the disk, along with a graphical representation of the partitions. The text area at the bottom of the screen displays all of the available devices and partitions. Selecting a device or partition at the bottom of the screen changes the display at the top of the screen.

The description of each partition shown in the text window at the bottom of the screen contains up to seven pieces of information:

Device The Device field contains the device name of the partition. For example, /dev/hda1 is the first partition on the first IDE hard drive.

Start The Start field contains the cylinder number of the physical location on the disk drive where the partition begins.

End The End field contains the cylinder number of the physical location on the disk drive where the partition ends.

Size The Size field displays the total size of the partition in megabytes.

Type The Type field displays the filesystem type. Disk Druid allows you to create five different types of filesystems:

swap is a special type used only for the swap file.

ext2 is the native Linux filesystem.

ext3 is the native Linux filesystem with added support for filesystem journalizing. (Journalizing is important because it prevents data from being lost during a system crash, and speeds reboot and recovery after a crash.)

vfat is the Microsoft FAT32 filesystem with added support for long names. You shouldn't need to create vfat partitions with Disk Druid because even if Linux is going to share a disk with Microsoft Windows, Windows should already be on the disk when you install Linux. Linux is very tolerant of other operating systems, and many of the installation tools are built with the assumption that Windows will be there when Linux is installed. Windows, on the other hand, is not designed to fit itself into the space left by other operating systems. Things run most smoothly if you let Windows take what it wants first and then install Linux.

software RAID is a special partition type used to create Redundant Arrays of Inexpensive Disks (RAID). See below for the Make RAID command.

Mount Point The Mount Point field contains the directory name under which this device is mounted. In Figure A.1, device /dev/hda1 is mounted as /boot, and /dev/hda2 is mounted as /. Notice that the swap partition does not have a traditional mount point. The swap space is not treated as a normal Linux directory.

Format The Format field indicates with a Yes or No whether or not the installation program should format the partition.

In the middle of the screen, between the text window at the bottom and the graphical representation at the top, Disk Druid displays the buttons that select the action you wish to take. The five buttons are

New, which adds a new partition.

Edit, which edits the selected partition.

Delete, which deletes an existing partition.

Reset, which removes any changes you made to the partition table during this run of Disk Druid.

Make RAID, which combines software RAID partitions into a RAID device. Before using this button, you must first use the New button to create partitions with a filesystem type of software RAID. To find out more about RAID, see *Linux System Administration* by Stanfield and Smith, Sybex 2000.

Using these buttons you can add, edit, or delete a partition. To delete a partition, for example, highlight the partition in the partition list at the bottom of the screen and then click Delete.

Deleting a Partition

Deleting a partition is frequently the first step in partitioning a disk; regardless of the type of system being installed, you usually start by removing unwanted partitions from the disk to make room for the new partitions. For example, new computers often come with Windows installed in a partition that consumes the whole disk. For a server installation, you start by deleting the Windows partition because servers do not normally dual–boot.

If you are installing Linux on a client system that will be dual–booting, you don't delete the Windows partition, but you do start by deleting the other partition created by fips or Partition Magic. (See the discussion of fips earlier in this appendix.)

Adding a Partition

To add a new partition to a disk, simply click on the New button in the Disk Druid main screen. This displays the box shown in Figure A.2. Here, you enter the mount point, select the filesystem type, and define the size of the new partition in megabytes.

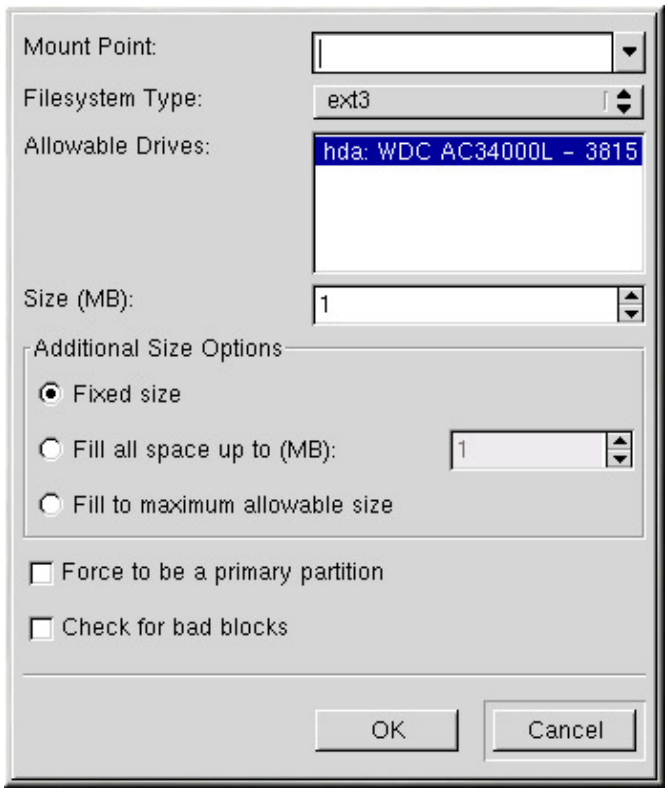


Figure A.2: Adding a partition in Disk Druid

Use the Allowable Drives list to select the disk drive on which to create the new partition. Make sure that only one drive is selected. Otherwise, Disk Druid can place the new partition on any one of the selected drives, which may not be where you intended.

Three check boxes control the size of the partition. You can enter a fixed size for the partition, allow the partition to grow to a specified size, or allow the partition to grow into all unused space on the disk. Most system administrators create partitions with a fixed size because they have anticipated reasonable growth, and want to ensure that the partition has that amount of storage allocated to it. Others prefer to let the partitions grow based on system demands. Both techniques work. The decision is control versus flexibility. Choose the approach that makes you most comfortable.

There is a check box to specify whether or not the installation program should check for bad blocks when formatting the partition. Checking for bad blocks is a good idea the first time the partition is formatted. It is usually unnecessary during subsequent formats.

Finally, there is a check box that allows you to force the new partition onto a primary partition. A disk can contain four primary partitions; all other partitions are logical partitions. (The different types of partitions are discussed later in this appendix, when we cover fdisk.) Forcing the partition to a primary partition is necessary only if the partition will contain the LILO boot loader. Chapter 1

explains how the boot loader is used. If your disk drive contains no more than four partitions, they can all be primary partitions.

Using Disk Druid to delete old partitions, add new ones, and edit existing ones, you can develop the partition table that you want. When you're done, click OK in the main window to exit Disk Druid. Alternatively, the partition table can be developed with and written to the disk using the traditional fdisk command.

Partitioning with *fdisk*

fdisk is a command–driven, text–based utility. At the fdisk prompt, enter any of the utility's single–character commands. Use the m command to display the list of commands shown here in Table A.2.

Table A.2: Single–Character *fdisk* Commands

Command	Function
A	Toggles the boot flag
B	Edits a BSD disk label
C	Toggles the DOS compatibility flag
D	Deletes a partition
L	Lists the partition types
M	Displays a list of fdisk commands
N	Adds a new partition
P	Prints the partition table
Q	Quits without saving changes
S	Create an empty label for a Sun Microsystems disk
T	Sets the partition type
U	Selects either sectors or cylinders as the units used to display partition size
V	Verifies the new partition table
W	Writes the partition table to the disk and exits
X	Enters experts mode

To partition a disk using fdisk, start by displaying the current partition table with the p command. Next, use the d command to delete any unneeded partitions. On a server, the unneeded partition is probably the Windows partition or the partition created by fips. Listing A.3 shows these initial steps:

Listing A.3: Partitioning with *fdisk*

```
# fdisk /dev/hda

Command (m for help): p

Disk /dev/hda: 240 heads, 63 sectors, 557 cylinders
Units = cylinders of 15120 * 512 bytes

   Device Boot Start  End    Blocks   Id  System
/dev/hda1          1  224   1693408+   c  Win95 FAT32 (LBA)
/dev/hda2        225  557   2517480    c  Win95 FAT32 (LBA)

Command (m for help): d
```

```
Partition number (1-4): 2
```

After making sufficient space on the hard drive for the Linux installation, create the Linux partitions with the n command. This example creates a new partition:

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 2
First cylinder (225-557, default 225): 225
Last cylinder or +size or +sizeM or (225-557, default 557): +128M
```

When you create a new partition, fdisk asks what type of partition you want to create, the specific cylinder where the partition will start, and the size of the partition. Start the partition at the first free cylinder. Set the size of the partition, either by selecting an ending cylinder for the partition or by using an absolute size in megabytes. In the example, we use +128M for a 128–megabyte partition. Setting the starting point and the size of a partition are easy; understanding partition types is a little more difficult.

Primary partitions and extended partitions are *physical partitions* because they exist as physical entities on the disk. A primary partition is always treated as a single entity. If partition number 2 is a primary partition, it can only be referred to as partition number 2. An extended partition is a physical partition that acts as a host for *logical partitions*. Logical partitions are logical entities that provide a way of referring to different parts of a physical partition. The preceding example shows primary partition number 2 being created. The first four partition numbers, partitions 1 to 4, are reserved for physical partitions. To create more than four partitions, first create an extended partition and then create logical partitions within that physical partition. Listing A.4 shows an example:

Listing A.4: Adding Logical Partitions

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
e
Partition number (1-4): 3
First cylinder (243-557, default 243): 243
Last cylinder or +size or +sizeM or +sizeK (243-557, default 557): 557
Command (m for help): n
Command action
   l   logical (5 or over)
   p   primary partition (1-4)
l
First cylinder (243-557, default 243): 243
Last cylinder or +size or +sizeM or +sizeK (243-557, default 557): 350

Command (m for help): n
Command action
   l   logical (5 or over)
   p   primary partition (1-4)
l
First cylinder (351-557, default 351): 351
Last cylinder or +size or +sizeM or +sizeK (351-557, default 557): 557
Command (m for help): p
```

```
Disk /dev/hda: 240 heads, 63 sectors, 557 cylinders
Units = cylinders of 15120 * 512 bytes

   Device Boot Start  End    Blocks   Id  System
/dev/hda1 *        1  224   1693408+   c  Win95 FAT32 (LBA)
/dev/hda2        225  242    136080   83  Linux
/dev/hda3        243  557   2381400    5  Extended
/dev/hda5        243  350    816448+  83  Linux
/dev/hda6        351  557   1564888+  83  Linux
```

In this example, we create an extended partition (hda3). After we have created an extended partition, subsequent fdisk n commands ask if we want to create a logical partition or a primary partition. In the example, we create two logical partitions. fdisk automatically assigns partition numbers to logical partitions, starting with partition number 5. The p command lists all of the partitions. Notice that the physical partition hda3 and the logical partition hda5 start at exactly the same physical location—cylinder 243. Further, notice that the physical partition ends at exactly the same cylinder—557—as the logical partition hda6. This view shows that extended partitions are really a way of subdividing physical partitions into additional parts.

I normally limit the partition table to the primary partitions. Four partitions per disk are generally enough; too many partitions create an unnecessarily complex structure. If I want more partitions, I usually add additional disks. The only real exceptions to this are when working with very large disks (60GB or more) that might benefit from a large number of partitions, or when using Disk Druid to create partitions that are allowed to grow. Disk Druid uses extended partitions to implement "growable" partitions.

After creating the new partitions, use the p command to view them. You'll notice in the example in Listing A.4 that the filesystem type for all newly created partitions is "Linux." Use the t command to set the correct partition types. The 128MB partition in the example is supposed to be a Linux swap partition. As shown in Listing A.5, a Linux swap partition is a type 82 partition:

Listing A.5: Assigning Filesystem Types

```
Command (m for help): t
Partition number (1-4): 2
Hex code (type L to list codes): 82
Changed system type of partition 2 to 82 (Linux swap)

Command (m for help): p

Disk /dev/hda: 240 heads, 63 sectors, 557 cylinders
Units = cylinders of 15120 * 512 bytes

   Device Boot Start  End    Blocks   Id  System
/dev/hda1 *        1  224   1693408+   c  Win95 FAT32 (LBA)
/dev/hda2        225  242    136080   82  Linux swap
/dev/hda3        243  557   2381400    5  Extended
/dev/hda5        243  350    816448+  83  Linux
/dev/hda6        351  557   1564888+  83  Linux
```

Filesystem types are entered as hex codes. The l command lists the hex value for each filesystem type. fdisk supports 85 different filesystem types! However, you rarely need to use most of them. A normal Linux server installation uses only the Linux swap filesystem (type 82) and the Linux filesystem (type 83).

The completed partition table is written to the disk with the w command.

```
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
Reboot your system to ensure the partition table is updated.

WARNING: If you have created or modified any DOS 6.x
partitions, please see the fdisk manual page for additional
information.
```

fdisk then terminates with a series of status messages. The final warning message is always printed. It tells you to create DOS partitions with the DOS fdisk program and Linux partitions with the Linux fdisk program. This is one reason I recommend installing Microsoft Windows first on a dual–boot client system. Then, use fips or Partition Magic to adjust the DOS partition before installing Linux.

As you can see, fdisk differs from Disk Druid in several ways. Disk Druid is used on Red Hat systems; fdisk runs on all versions of Linux. fdisk requires more manual input than does Disk Druid. Additionally, when you create a partition, fdisk does not allow you to assign the mount point to the partition as Disk Druid does. You must define the mount points separately. Every version of Linux gives you some way to define mount points for the partitions.

Whether you enter the mount points in Disk Druid or later in the installation process, mount points are stored in the filesystem table, fstab, which is read during the boot. The fstab file is covered in Chapters 1 and 9.

Installing the Boot Loader

After you have configured the partitions, the Red Hat installation asks you to select a boot loader, to decide where you want it installed, and to select which partitions should be booted by the loader. Red Hat 7.2 offers two boot loaders: GRUB and LILO. GRUB is new for Red Hat in 7.2; LILO is much more widely used by other Linux distributions. Most people who install Red Hat 7.2 use GRUB because it is the default. I use LILO because it is what I'm used to, and it is what I use on my other Linux systems. However, GRUB works just as well, and if GRUB is selected, the Red Hat installation program gives you a chance to enter a boot password. A boot password controls who can provide boot prompt input when the system reboots, which enhances security. Both GRUB and LILO have password capability, but Red Hat makes it easy to enter a password for GRUB. The detailed configuration of both GRUB and LILO is covered in Chapter 1.

Regardless of which boot loader you choose, the boot loader can be installed either in the master boot record (MBR) or in the first sector of the boot partition. The MBR is the first sector of a disk. The boot partition is the partition marked "active" or "bootable" in the partition table.

There are advantages and disadvantages to both locations. If the boot loader is installed in the MBR, anything in the existing MBR is destroyed. Most of the time, this is unimportant. Linux boot loaders are designed to be compatible with the standard function and structure of the master boot record. But on rare occasions, things can go wrong when you write over the MBR. For example, I once lost the use of an old hard drive by installing LILO on the master boot record and accidentally overwriting the Western Digital EZ Drive software stored there. To recover, I had to reinstall the EZ Drive software and then install LILO in the first sector of the Linux boot partition.

If the boot loader is installed in the first sector of the boot partition, the MBR is unaffected, but another problem may appear. The problem occurs when the boot partition is a logical partition, because a standard MBR only boots an active physical partition. To avoid this problem when working with logical partitions, install the boot loader in the master boot record.

To put the boot loader in the first sector of the boot partition, place the Linux boot partition in a physical partition. With Linux in a physical partition, you can install the loader in either the MBR or the boot sector of the root partition.

The Red Hat installation program gives you a chance to enter any required kernel parameters. This is the same thing as the boot prompt input mentioned earlier in this appendix and described in Chapter 1. Most systems do not require any input at this time, but if you had to use boot input in order to run the Linux installation program, you may need to use that same input *every* time Linux boots. See Chapter 1 for more information about boot input, GRUB, and LILO.

At the bottom of the screen, the installation program displays a list of all of the partitions that it thinks you might want to boot. It will list the Linux boot partition and assign it a boot label. Most distributions use the label "linux"; Red Hat 7.2 uses the label "Red Hat Linux." The Default check box shows which partition the loader will boot by default.

If the disk has a Windows partition, the Red Hat installation program assigns it the boot label "DOS." You can change the boot labels or change the default boot partition during the installation. Generally, there is no reason to do this.

Configuring the Ethernet Adapter

Configuring the network interface is the next step in the Red Hat 7.2 installation process. If this is a system that was installed via the network, you're given the option of keeping the temporary configuration created for the installation. For all other systems, the network interface is configured at this time.

Note Details of the network configuration created by the installation program are covered in Chapter 2, "The Network Interface."

The Red Hat installation program provides two configuration techniques: DHCP (Dynamic Host Configuration Protocol) or manual configuration. Client systems can use DHCP. If you are configuring a desktop, simply select "Configure using DHCP" and the network configuration is complete, because the DHCP server provides all of the configuration information. To use DHCP, you must, of course, install, maintain, and operate a DHCP server. Chapter 8, "Desktop Configuration Servers," describes how to set up a DHCP server.

A Linux network server, however, is not usually configured by DHCP—a network server needs a dedicated address. To configure a server, manually enter an IP address, a network mask, a network address, a broadcast address, a hostname, a default gateway address, and up to three DNS server addresses. The Red Hat installation program makes its best guess for several of the configuration values based on the IP address you enter. Check these values to make sure they are what you really want. The values for all of these configuration parameters should be the values you decided on during the installation planning described earlier.

Configuring the Firewall

The 2.4 Linux kernel implements kernel–level packet filtering that is configurable with the iptables command. The iptables command and how it is used to create a basic firewall are covered in Chapter 12, "Security." Red Hat 7.2 includes firewall configuration during the installation. The installation program does not provide the same level of control as you get from building your own iptables rules (as described in Chapter 12), but it does provide a simple way to create a basic access control packet filter. The Red Hat firewall configuration screen is shown in Figure A.3.

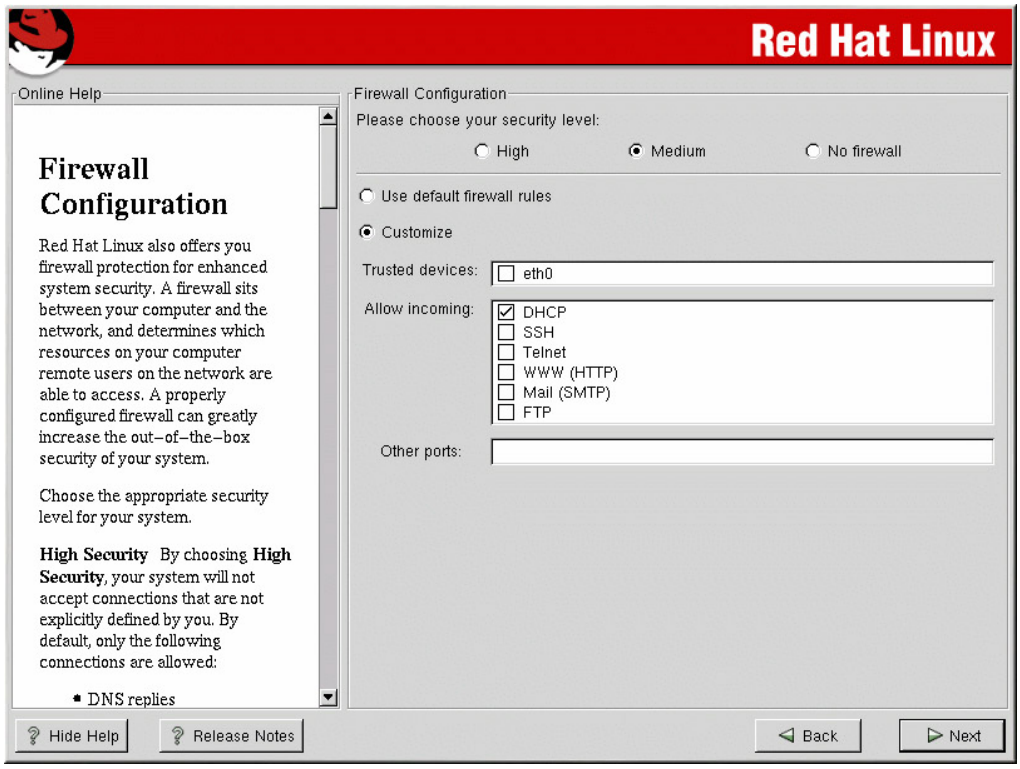


Figure A.3: Red Hat firewall configuration

At the top of the screen are three security levels:

High The High security level blocks all incoming connections. This does not prevent the local system from connecting to services offered by remote systems, and it does not block DNS or DHCP replies. This is a good setting for desktop clients because the desktop user can access remote services, but no one can access a service on the desktop that the user may have accidentally left running.

Medium The Medium security level blocks any incoming connections bound for ports below 1024. Port numbers below 1024 are called *privileged ports*, and are assigned to well–known services. Many of them are the services listed in the /etc/services file. Additionally, connections to the NFS server port, the X Windows System ports, and the X Font server port are blocked. Connections to most non–privileged ports are allowed. This is a good setting for desktop systems that use tools such as RealAudio, which require inbound connections to non–privileged ports.

No firewall The No firewall setting allows all incoming connections, providing no packet filtering. This setting is used when you plan to create your own firewall configuration with iptables after the installation is complete.

The default port–filtering restrictions associated with the High and Medium security settings can be eased by clicking the Customize button. When the Customize button is selected, three areas become active on the screen:

Trusted devices The Trusted device box lists all network interfaces configured on the system. Clicking the check box next to a listed interface means that the firewall packet filters will not be applied to any traffic received from this device. This is used only on multihomed systems; that is, computers connected to more than one network. It allows you to apply filters to the connection to the outside world, while allowing the internal network to have unlimited access to the system. In Figure A.3, there is only one Ethernet interface, eth0, configured. In this case, the Trusted device box would not be used.

Allow incoming The Allow incoming box provides check boxes for six services: DHCP, SSH, Telnet, HTTP, SMTP, and FTP. By default, inbound connections for all six of these services are denied. Checking a box allows inbound connections to that service.

Other ports The Other ports box is used to enter other ports for which incoming connections should be allowed. Ports are entered in the box in the format of ***port:protocol***, where ***port*** is a port number or the name of a service from the /etc/services file, and ***protocol*** is a protocol name from the /etc/protocols file. (***protocol*** is usually either tcp or udp.) For example, to allow incoming connections to an imap server, you would enter **imap:tcp** in this box.

The default High and Medium security settings are used for desktop client systems. Servers must allow incoming connections to the services that they offer. A server might start with a High security setting, but it then must be customized to reduce that level of security. For example, an e–mail server might start with the High setting, but then Customize would be used to allow incoming SMTP connections, and perhaps to allow connections to the POP3 and IMAP services. Generally, network server administrators select No firewall on this screen and then build a firewall using iptables, as described in Chapter 12.

After configuring the firewall, the installation asks you to select the language that should be used and to set the system clock. You are then asked to enter a root password. Selecting the root password during the initial installation ensures that you don't forget to select one before the system comes online. Any account without a password is an open invitation to an intruder. A root account without a password is a security disaster. Be sure to pick a good password (there are guidelines on how to pick a good one in Chapter 12).

You're also given the opportunity to create user accounts. This is not the time to create a bunch of user accounts, but you should create at least one account for yourself. You'll use that account later to log in to the system to finish customizing the server configuration.

Selecting an Authentication Type

In addition to selecting a root password, the installation program gives you the opportunity to select optional password security. Traditionally, Linux stores passwords in the /etc/passwd file, and it encrypts the passwords stored there using standard Unix password encryption. There are two problems with this. First, Unix password encryption uses only the first eight characters of the password. Limiting passwords to eight characters limits a user's choice of passwords, and makes them easier to guess. Second, the /etc/passwd file must be "readable" by every user and process

on the system. Storing encrypted passwords in a file that everyone can read makes them vulnerable to attacks with password–guessing programs.

Two check boxes at the top of the Authentication Configuration window address these security problems. Select the Enable Shadow Passwords box to store the encrypted passwords in the /etc/shadow file, which cannot be read by all users. Next, select the Enable MD5 Passwords check box to use the Message Digest 5 (MD5) algorithm to encrypt your passwords. MD5 encryption allows for passwords up to 256 characters in length, far larger than any user will ever need. Figure A.4 shows the Authentication Configuration window with both of these security enhancements selected.

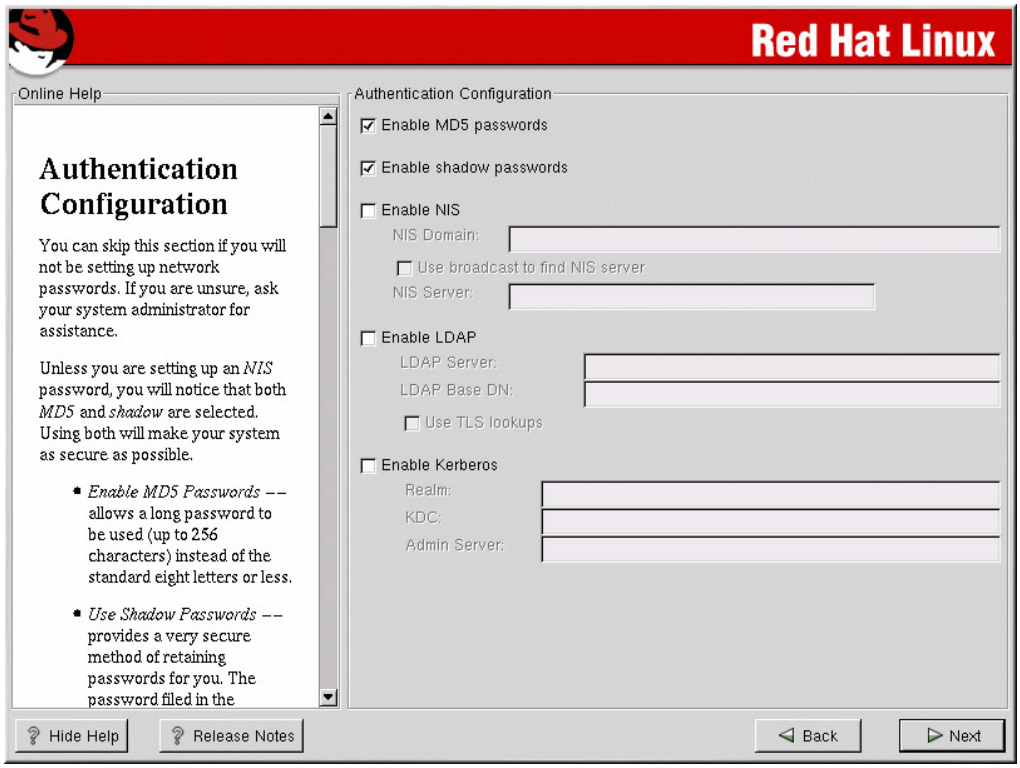


Figure A.4: The Authentication Configuration screen

In addition to these two security enhancements, the Authentication Configuration window can be used to select a password server. There are four types of password servers available:

NIS Originally developed by Sun Microsystems, Network Information Service (NIS) stores several administrative databases on a networked server. These databases include / etc/passwd and /etc/shadow. If you use a NIS server for passwords, select Enable NIS, and provide your network's NIS domain name and how the NIS server should be located—by name or by broadcast.

LDAP The Lightweight Directory Access Protocol (LDAP) is a directory service that can be configured to provide a wide range of information, including passwords. If you obtain passwords from an LDAP server, enable LDAP, enter the server name, and enter the distinguished name of the directory to be used. If the LDAP service is not running directly on your server, you can use Transport Layer Security (TLS) to secure the connection to the remote server.

Kerberos 5 Kerberos is a network–based authentication service originally developed at MIT. If your system uses a Kerberos server to authenticate users, enable Kerberos, and enter the name of your Kerberos realm and the name of the server. The name of the authentication server goes in the KDC box. (KDC stands for

Key Distribution Center.) If you are allowed to access the Kerberos administration server, you can also enter the name of that server in the Admin Server box.

SMB Server Message Block (SMB) protocol is used on Microsoft networks for file and printer sharing, and for authentication. If you use SMB for user authentication, enable SMB, and enter the name of your workgroup and the name of the SMB server. Configuring a Linux system to act as an SMB server is covered in Chapter 9, with additional information in Chapter 10, "Printer Services."

As you can see, Linux provides a range of possible user authentication techniques. User authentication is discussed in more detail in Chapter 12.

Installing the Software

The installation program asks you to select the software to install. All Linux distributions provide similar choices. You can select related groups of software (Red Hat calls them components), individual packages, or everything.

Installing everything is simple, but it is only useful when all functions from compute server to mail server are concentrated on one system. Installing everything is not the best approach for most operational servers. Unneeded software wastes disk storage, and can open a hole for an intruder to crawl through.

Selecting individual packages gives the most control over the installation, but it also requires the most knowledge about each piece of software—and it is very time–consuming. There are several hundred individual packages, and many of them are dependent on other packages to function properly. After you select the packages that should be installed, Red Hat checks for package dependencies, and warns you if a package needs additional software to function properly. At that time, you must allow the system to install the packages needed to satisfy the dependencies.

Selecting components is a good compromise for most systems. It is faster than reviewing individual packages for installation, and it gives you more control over the configuration than installing everything.

Select the minimum software needed to effectively run the server. Sometimes, this is not obvious. For example, you might plan on creating a dedicated mail server. You know you need sendmail and the TCP/IP network software, but what about programming languages such as C? You may want to download and compile the latest version of sendmail, which would require C. And you need m4 to create the sendmail configuration file. One possibility is to have a separate development machine compile sendmail, and build the configuration files before placing them on the mail server. This is a more secure configuration than having a compiler sitting on the mail server, but many system administrators prefer to have the compiler on the machine where it is needed. Select the packages that fit the way you manage your systems.

X Windows

Many Unix servers run with only a command–line interface, and do not use X at all. If a command–line interface is all that you want, don't select the X Windows components for the software installation. Many Linux servers, however, do use X. X provides a powerful console interface, and there are a number of X–based tools available for system administration.

The Red Hat installation program attempts to configure X if X Windows components were selected. The installation program probes the system, and selects a video card. You can accept the card selected by the installation program, manually enter your own X configuration, or skip X configuration.

Note Red Hat 7.2 first configures the video card and then does unrelated tasks (installing software from the CD–ROM and creating the boot disk) before returning to the monitor configuration. Despite this, video card configuration and monitor configuration are both components of X configuration.

If you decide to configure X manually, make sure you know the horizontal and vertical sync ranges of your monitor, the monitor's maximum resolution, the make and model of your video card, the amount of memory on the video card, and whether or not the video card has a clock chip (and if it does, the model of that chip). Armed with this information, you'll have no problems configuring X.

If you don't have all of the required information at your fingertips, accept the default values provided by Red Hat. They should work, and you're given a chance to test them before making a final decision. Figure A.5 shows the final window of the X configuration. In this window, select the desired screen resolution, and click Test Setting. If the display is to your liking, accept the settings. If you do not get a good display, you can try other settings, or skip X configuration for now. To skip X configuration, select Text as the logon type. Many Linux system administrators put off configuring X Windows until later.

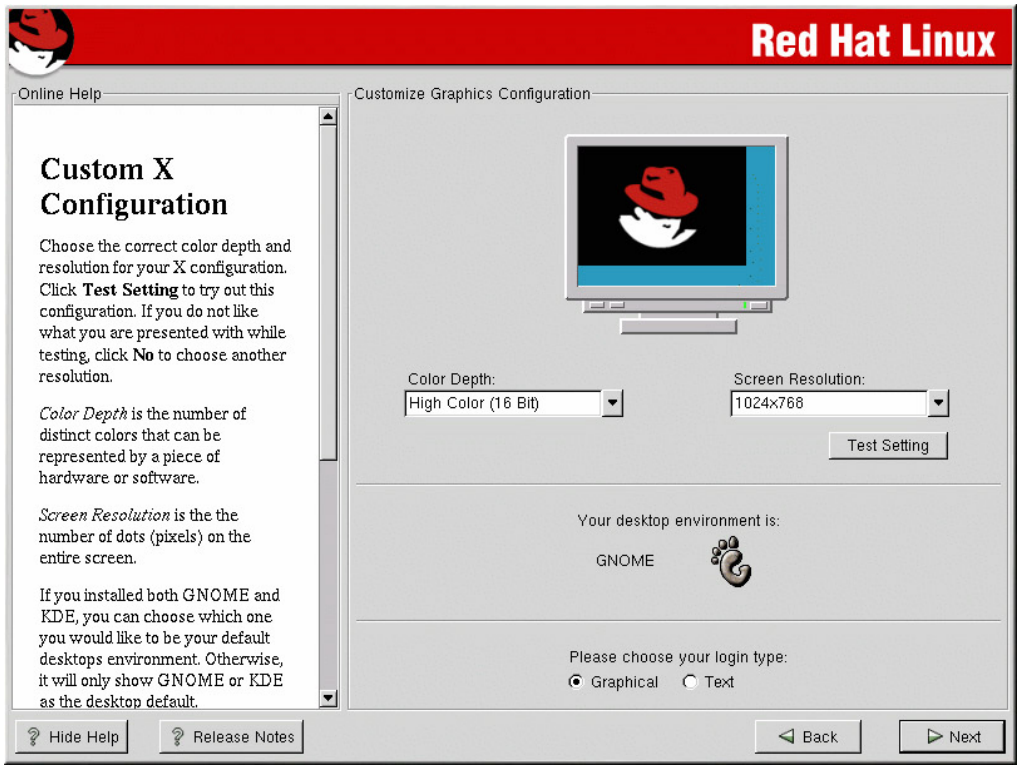


Figure A.5: Final X configuration window

X configuration can be complex. If you have trouble configuring X, don't worry—you can boot the system with no problems because X is not essential to get the system running. If you select Text as the logon type, you can boot the system, log on at the text prompt, and run Linux from the text command line. Later, when you're ready to configure X, run one of the X configuration tools.

Using *Xconfigurator*

Xconfigurator is the program that the Red Hat installation uses to configure X Windows. It can be run at any time by typing **Xconfigurator** at the shell prompt. When it is run from the shell prompt, Xconfigurator provides a menu–driven graphical interface. Use the tab and the arrow keys to move around the menu, and the spacebar and enter key to select items.

When Xconfigurator starts, it displays an introductory window. Select OK to start the configuration. Xconfigurator then probes the system to detect the video card and the monitor configuration. It displays the results of each probe on separate screens. If they are correct, accept those configuration values. (If the values are not correct, rerun Xconfigurator in ––expert mode, as described later.)

Next, Xconfigurator asks how much memory is available on the video card. Enter the correct amount. It then asks what clock chip is used on the video card. Either select the correct clock chip, or select No Clockchip, which is the recommended setting.

Finally, you're asked to select the video mode, which is the combination of the color depth and the screen resolution. Xconfigurator determines the possible video modes from the information that has been provided so far in the configuration. Most systems offer three color depths (8–, 16–, and 24–bit), and various screen resolutions for each color depth. Select the resolution you want from the display, and select OK. Xconfigurator restarts X, and tests the new configuration. If it is what you want, you're done. If not, select another resolution until you get the display you want.

The preceding description assumes that Xconfigurator can determine the correct configuration of your system through probing, which may not be the case. To have more configuration options, run Xconfigurator with the ––expert command–line argument. The system still probes the adapter and the monitor, but it also provides a very long list of supported video cards and preconfigured monitors. All you need to do is select your video card from the first list and your monitor from the second list. Additionally, ––expert mode gives you the opportunity to manually select the characteristics of a video card and those of a monitor if your video card and monitor are not in the lists of supported hardware.

Running Xconfigurator with the ––expert option after Linux has successfully booted can help you configure X, even if you could not successfully configure it during the initial installation. In most cases, X is easily configured when Linux is installed, but Xconfigurator is there if you need it.

The Boot Floppy

As noted earlier, X configuration is not a contiguous process during the initial Red Hat installation. Between the selection of the video card and the configuration of the monitor, the Red Hat installation program asks you to create a boot floppy for the new system. This is not the same as the installation boot floppy that was created earlier in this appendix. The floppy you create during this step contains the Linux kernel and the basic configuration files needed to boot a fully operational server with the configuration you have just created. It is absolutely essential.

Warning Don't skip this step! The boot disk is essential if something goes wrong.

You're done! Red Hat warns you that it will reboot the system; when it does, the system should boot up under Linux. Chapter 1 covers the boot process in detail.

In Sum

Most of the installation steps in this appendix are integrated into a single installation program. Nonetheless, each step is distinct, and requires your thought and input. Proper planning is essential for everything to go smoothly. Most of the decisions in an installation hinge on knowing the hardware upon which the server is built and the software needed for the server to fulfill its purpose. Understand the details of the server's hardware, and know exactly what you want to use the server for before you begin the installation. Develop a partition plan suitable for the server's purpose. With this preparation, you'll be ready for the installation.

This appendix contains a detailed example of installing Red Hat Linux 7.2. However, there are many different Linux distributions to choose from. If you have time, and I know that is a big *if* for a network administrator, pick a few distributions and work with them. Installing some different distributions provides insight into the workings of Linux and increased confidence in dealing with Linux problems. A side−by−side comparison helps you select the Linux you like best. Linux distributions are so inexpensive that money is not the problem with this approach. The problem is, of course, time.

Furthermore, use a distribution that comes with high−quality installation documentation. Buy the distribution's boxed set that includes the full set of CD−ROMs and a printed installation guide. Installation procedures can change dramatically from one release to another, and the most current documentation is your best protection against being caught off−guard by these changes. Only the documentation that comes with the distribution can give you all the details you need to install the distribution you select.

Appendix B: BIND Reference

Overview

There are two versions of the Berkeley Internet Name Domain (BIND) software in widespread use. Current versions of Linux ship with BIND 9; recent versions of Linux shipped with BIND 8. The DNS database files used by these different versions of BIND are the same. The most apparent changes in the two versions of BIND are a few new options and statements in the configuration files.

This appendix provides a reference to the syntax and structure of the configuration commands for both BIND 9 and BIND 8. It is a reference, not a tutorial. (See Chapter 4, "Linux Name Services," for a tutorial on configuring a Linux DNS server and for realistic examples of the configuration commands that you will actually use.) Most of the commands shown here are not required for an average installation; some of these commands are useful only for root domain servers. Use this appendix to help you read unfamiliar commands in a sample configuration file. Use Chapter 4 to help you create your own configuration file.

The syntax of the BIND configuration commands is described with these conventions:

bold Indicates that something must be typed as shown.

italic Indicates that you provide your own value for the specified field.

Square brackets [ ] Indicates that the item is optional.

Vertical bar | Indicates that you choose one keyword or the other.

Note The source for much of this information is the online documentation at the http://www.isc.org/ website. Visit that website for the latest information.

named.conf Commands

The named.conf file is used to configure the BIND name server daemon. The file defines the named operational environment, and points to the sources of DNS database information. For BIND 8, the file is composed of eight basic configuration statements: acl, options, logging, zone, server, key, trusted–keys, and controls. BIND 9 uses the same eight statements and adds a ninth— the view statement. In addition to these configuration commands, an include statement can be used with both BIND 8 and BIND 9. include causes an external file to be loaded into the named.conf file. The external file can contain any or all of the basic configuration statements.

The *options* Statement

The options statement defines global options that affect the operation of BIND and the DNS protocol. The syntax of the options command for BIND 8 is shown in Listing B.1:

Listing B.1: The BIND 8 *options* Statement Syntax

```
options {
  [ version string; ]
  [ directory pathname; ]
  [ named-xfer pathname; ]
  [ dump-file pathname; ]
  [ memstatistics-file pathname; ]
```

```
  [ pid-file pathname; ]
  [ statistics-file pathname; ]
  [ auth-nxdomain yes|no; ]
  [ deallocate-on-exit yes|no; ]
  [ dialup yes|no; ]
  [ fake-iquery yes|no; ]
  [ fetch-glue yes|no; ]
  [ has-old-clients yes|no; ]
  [ host-statistics yes|no; ]
  [ multiple-cnames yes|no; ]
  [ notify yes|no; ]
  [ recursion yes|no; ]
  [ rfc2308-type1 yes|no; ]
  [ use-id-pool yes|no; ]
  [ treat-cr-as-space yes|no; ]
  [ also-notify { address-list; }; ]
  [ forward only|first; ]
  [ forwarders { address-list; }; ]
  [ check-names master|slave|response  warn|fail|ignore; ]
  [ allow-query { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ allow-recursion { address_match_list }; ]
  [ blackhole { address_match_list }; ]
  [ listen-on [ port ip_port ] { address_match_list }; ]
  [ query-source [address ip_addr|*] [port ip_port|*] ; ]
  [ lame-ttl number; ]
  [ max-transfer-time-in number; ]
  [ max-ncache-ttl number; ]
  [ min-roots number; ]
  [ serial-queries number; ]
  [ transfer-format one-answer|many-answers; ]
  [ transfers-in  number; ]
  [ transfers-out number; ]
  [ transfers-per-ns number; ]
  [ transfer-source ip_addr; ]
  [ maintain-ixfr-base yes|no; ]
  [ max-ixfr-log-size number; ]
  [ coresize size; ]
  [ datasize size; ]
  [ files size; ]
  [ stacksize size; ]
  [ cleaning-interval number; ]
  [ heartbeat-interval number; ]
  [ interface-interval number; ]
  [ statistics-interval number; ]
  [ topology { address_match_list }; ]
  [ sortlist { address_match_list }; ]
  [ rrset-order { order_spec ; [ order_spec ; ... ] ] };
};
```

The meanings of some of the values used with the options are fairly obvious. *pathname* is a file path, which is either relative to the value provided for the directory option or an absolute path from the root. *ip_port* is an IP port number; *number* is just that, a number; and *size* is the size of a file in bytes, which can be abbreviated with K for kilobyte or M for megabyte (for example, 5M is 5 million bytes). Boolean values are yes or no, and keyword values are the keywords shown in Listing B.1 (for example,. one–answer is a keyword value that must be typed as shown in the syntax).

Several options accept IP addresses as arguments. *ip_addr* is a single IP address. An *address–list* is a list of IP addresses, separated by semicolons. An address in an *address_match_list* can include an optional address mask prefix. For example, 172.16.0.0/16 matches every address in which the

first 16 bits are 172.16. An exclamation mark (!) before an address means "don't match" the value. Therefore, placing an exclamation mark in front of our sample address would indicate to match everything except addresses in which the first 16 bits are 172.16. An *address_ match_list* can also include special keywords:

any Matches every possible address.

none Matches no addresses.

localhost Matches every address assigned to the local host.

localnet Matches every address where the network portion of the address is the same as the network portion of any address assigned to the local host.

The *order_spec* value has a special syntax of its own. An *order_spec* is a rule that defines how to order resource records when multiple records are sent in response to a single query. The structure of an *order_spec* is

```
[class class ][ type type ][ name "domain" ] order order
```

where *class*, *type*, and *domain* are the values found in the class, type, and name fields of the resource records to be sorted, and *order* is one of three possible values:

fixed The order in which records are defined in the zone file is maintained.

random Resource records are shuffled into a random order.

cyclic The resource records are rotated in a round−robin manner, which is the default order.

Each configuration option available for BIND 8 is described in Table B.1.

Table B.1: BIND 8 Configuration Options

Option	Meaning
version	The string returned when the server is queried for its version.
directory	The path of the working directory from which the server reads and writes files.
named−xfer	The path to the named−xfer program.
dump−file	The file where the database is dumped if named receives a SIGINT signal. The default is named_dump.db.
memstatistics−file	The file where memory usage statistics are written if deallocate−on−exit is set. The default is named.memstats.
pid−file	The file where the process ID is stored.
statistics−file	The file where statistics are written when named receives a SIGILL signal. The default is named.stats.
auth−nxdomain	yes causes the server to respond as an authoritative server. The default is yes.
deallocate−on−exit	yes is used to detect memory leaks. The default is no.
dialup	

	yes optimizes the server for a dial−up network operation. The default is no.
fake−iquery	yes causes the server to respond to inverse queries with a fake reply instead of an error. The default is no.
fetch−glue	yes causes the server to fetch all of the glue records for a response. The default is yes.
has−old−clients	yes sets auth−nxdomain and maintain−ixfr−base to yes and rfc2308−type1 to no for backward compatibility.
host−statistics	yes causes the server to keep statistics on every host. The default is no.
multiple−cnames	yes allows multiple CNAME records for a domain name. The default is no.
notify	yes, the default, causes the server to send DNS NOTIFY messages when a zone is updated.
recursion	yes, the default, causes the server to recursively seek answers to queries.
rfc2308−type1	yes returns NS records with the SOA record for negative caching. no, the default, returns only the SOA record for compatibility with old servers.
use−id−pool	yes tracks outstanding query IDs to increase randomness. no is the default.
treat−cr−as−space	yes treats carriage returns as spaces when loading a zone file. no is the default.
also−notify	Identifies unofficial name servers to which the server should send DNS NOTIFY messages.
forward	first causes the server to first query the servers listed in the forwarders option and then look for the answer itself. only causes the server to query only the servers listed in the forwarders option.
forwarders	Lists the IP addresses of the servers to which queries are forwarded. The default is not to use forwarding.
check−names	Checks hostnames for compliance with the RFC specifications. You can select to check names when the master server loads the zone (master), when the slave transfers the zone (slave), or when a response is processed (response). If an error is detected, you can choose to ignore it (ignore), send a warning to the administrator about it (warn), or reject the bad name (fail).
Option	Meaning
allow−query	Queries are accepted only from hosts in the address list. The default is to accept queries from all hosts.
allow−transfer	Only hosts in the address list are allowed to receive zone transfers. The default is to allow transfers to all hosts.
allow−recursion	Only listed hosts are allowed to make recursive queries through this server. The default is to do recursive queries for all hosts.
blackhole	Lists hosts from which this server will not accept queries.
listen−on	Defines the interfaces and ports on which the server provides name service. By default, the server listens to the standard port (53) on all installed interfaces.
query−source	Defines the address and port used to query other servers.

lame−ttl	Sets the amount of time a lame server indication is cached. The default is 10 minutes.
max−transfer−time−in	Sets the maximum amount of time the server waits for an inbound transfer to complete. The default is two hours.
max−ncache−ttl	Sets the amount of time this server caches negative answers. The default is three hours, and the maximum acceptable value is seven days.
min−roots	Sets the minimum number of root servers that must be reachable for queries involving the root servers to be accepted. The default is 2.
serial−queries	Sets the number of outstanding SOA queries a slave server can have at one time. The default is 4.
transfer−format	one−answer transfers one resource record per message. many−answers transfers as many resource records as possible in each message. For compatibility with older systems, the default is one−answer.
transfers−in	Sets the maximum number of concurrent inbound zone transfers. The default value is 10.
transfers−out	Limits the number of concurrent outbound zone transfers.
transfers−per−ns	Limits the number of concurrent inbound zone transfers from any single name server. The default value is 2.
transfer−source	Specifies the network interface this server uses to transfer zones from remote masters.
maintain−ixfr−base	yes logs incremental zone transfers. no is the default.
max−ixfr−log−size	Sets the maximum size of the incremental zone transfer log file.
coresize	Sets the maximum size of a core dump file.
datasize	Limits the amount of data memory the server may use.
files	Limits the number of files the server may have open concurrently. The default is unlimited.
stacksize	Limits the amount of stack memory the server may use.
cleaning−interval	Sets the time interval for the server to remove expired resource records from the cache. The default is 60 minutes.
heartbeat−interval	Sets the time interval used for zone maintenance when the dial−up option is set to yes. 60 minutes is the default.
interface−interval	Sets the time interval for the server to scan the network interface list looking for new interfaces or interfaces that have been removed. The default is 60 minutes.
statistics−interval	Sets the time interval for the server to log statistics. The default is every 60 minutes.
topology	Forces the server to prefer certain remote name servers over others. Normally, the server prefers the remote name server that is topologically closest to itself.
sortlist	Defines a sort algorithm applied to resource records before sending them to the client.
rrset−order	Specifies the ordering used when multiple records are returned for a single query.

There are many similarities between the BIND 8 and BIND 9 options statements. Many BIND 9

options are the same as those used for BIND 8, and perform exactly the same functions. But there are differences. Several BIND 8 options are not used with BIND 9. Yet the list of options is no shorter, because some new options have been added. The BIND 9 syntax of the options command is shown in Listing B.2:

Listing B.2: The BIND 9 *options* Statement Syntax

```
options {
  [ version string; ]
  [ directory pathname; ]
  [ additional-from-auth yes|no; ]
  [ additional-from-cache yes|no; ]
  [ dump-file pathname; ]
  [ pid-file pathname; ]
  [ statistics-file pathname; ]
  [ auth-nxdomain yes|no; ]
  [ dialup yes|no; ]
  [ notify yes|no|explicit; ]
  [ notify-source [ip_addr|*] [port ip_port] ; ]
  [ notify-source-v6 [ip_addr|*] [port ip_port] ; ]
  [ recursion yes|no; ]
  [ recursive-clients number; ]
  [ tcp-clients number; ]
  [ also-notify { address-list; }; ]
  [ forward only|first; ]
  [ forwarders { address-list; }; ]
  [ allow-notify { address_match_list }; ]
  [ allow-query { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ allow-recursion { address_match_list }; ]
  [ blackhole { address_match_list }; ]
  [ listen-on [ port ip_port ] { address_match_list }; ]
  [ listen-on-v6 [port ip_port ] { address_match_list };  ]
  [ port ip_port; ]
  [ query-source [address ip_addr|*] [port ip_port|*] ; ]
  [ query-source-v6 [address ip6_addr|*] [port ip_port|*] ;]
  [ lame-ttl number; ]
  [ max-transfer-time-in number; ]
  [ max-transfer-time-out number; ]
  [ max-transfer-idle-in number; ]
  [ max-transfer-idle-out number; ]
  [ max-refresh-time number; ]
  [ max-retry-time number; ]
  [ max-cache-ttl number; ]
  [ max-ncache-ttl number; ]
  [ min-refresh-time number; ]
  [ min-retry-time number; ]
  [ transfer-format one-answer|many-answers; ]
  [ transfers-in number; ]
  [ transfers-out number; ]
  [ transfers-per-ns number; ]
  [ transfer-source ip_addr|* [port ip_port|*] ;]
  [ transfer-source-v6 ip6_addr|* [port ip_port|*] ;]
  [ coresize size; ]
  [ datasize size; ]
  [ files size; ]
  [ stacksize size; ]
  [ cleaning-interval number; ]
  [ heartbeat-interval number; ]
  [ interface-interval number; ]
  [ sortlist { address_match_list }; ]
  [ sig-validity-interval number; ]
```

```
  [ tkey-dhkey key_name key_tag; ]
  [ tkey-domain domain; ]
  [ zone-statistics yes|no; ]
};
```

A few of the new options have been added to BIND 9 to handle IPv6, which is an integral part of BIND 9. These options, listen–on–v6, notify–source–v6, query–source–v6, and transfer–source–v6, perform exactly the same functions as the like–named options do for IPv4, with the exception that these options perform those functions for IPv6. Table B.2 shows the new BIND 9 options that apply to IPv4.

Table B.2: New BIND 9 Options

Option	Meaning
additional–from–auth	yes, the default, uses information from any zone for which the server is authoritative to complete the additional data section of a response.
additional–from–cache	yes, the default, uses cached information to complete the additional data section of a response.
notify–source	Defines the address and port used to send DNS NOTIFY messages.
recursive–clients	Defines the maximum number of outstanding recursive lookups the server will accept. Defaults to 1000.
tcp–clients	Defines the maximum number of concurrent client connections. Defaults to 1000.
allow–notify	Identifies the servers that are permitted to send DNS NOTIFY messages to the slave servers.
port	Defines the port number used by the server. Defaults to 53.
max–transfer–time–out	Defines the maximum time allowed for outbound zone transfers. The default is two hours.
max–transfer–idle–in	Defines the maximum idle time allowed for an inbound zone transfer. The default is 1 hour.
max–transfer–idle–out	Defines the maximum idle time allowed for an outbound zone transfer. The default is one hour.
max–refresh–time	Sets the maximum refresh time allowed when this server acts as a slave. This value overrides the refresh time set in the master server's SOA record.
max–retry–time	Sets the maximum retry time allowed when this server acts as a slave. This value overrides the retry time set in the master server's SOA record.
max–cache–ttl	Sets the maximum amount of time this server will cache data. This value overrides the TTL values defined in the zone from which the data was retrieved.
min–refresh–time	Sets the minimum refresh time allowed when this server acts as a slave. This value overrides the refresh time set in the master server's SOA record.
min–retry–time	Sets the minimum retry time allowed when this server acts as a slave. This value overrides the retry time set in the SOA record of the zone for which this server acts as a slave.

sig–validity–interval	Defines the amount of time that digital signatures generated for automatic updates are considered valid. The default is 30 days.
tkey–dhkey	Identifies the Diffie–Hellman key used by the server to generate shared keys.
tkey–domain	Defines the domain name appended to shared keys.
zone–statistics	yes causes the server to collect statistics on all zones. The default is no.

Options change over time. Check the documentation that comes with the BIND 9 distribution for the latest list of options.

The *logging* Statement

The logging statement defines the logging options for the server. The syntax of the logging command for BIND 8 is shown in Listing B.3.

Listing B.3: BIND 8 *logging* Command Syntax

```
logging {
  [ channel channel_name {
    ( file pathname
       [ versions number|unlimited ]
       [ size size ]
    |syslog (kern|user|mail|daemon|auth|syslog|lpr
                |news|uucp|cron|authpriv|ftp
                |local0|local1|local2|local3
                |local4|local5|local6|local7)
    |null;)

    [ severity critical|error|warning|notice
                 |info|debug [level]|dynamic; ]
    [ print-category yes|no; ]
    [ print-severity yes|no; ]
    [ print-time yes|no; ]
  }; ]

  [ category category_name {
    channel_name; [ channel_name; ... ]
  }; ]
  ...
};
```

The logging statement can include two different types of subordinate clauses: the channel clause and the category clause. The channel clause defines how logging messages are handled. Messages are written to a file (file), sent to syslogd (syslog), or discarded (null). If a file is used, you can specify how many old versions are retained (versions), and how large the log file is allowed to grow (size). If syslogd is used, select a syslogd facility, such as daemon or authpriv, to log the messages. For both types of logging, you can also specify the severity of the messages written to the log (severity), and that the time (print–time), category (print–category), and severity (print–severity) of the message be included in the log entry.

The category clause defines the category of messages sent to the channel. Thus, the category clause defines what is logged, and the channel clause defines where it is logged. The logging categories are listed in Table B.3.

Table B.3: BIND 8 Logging Categories

Category	Type of Messages Logged
cname	Messages recording CNAME references.
config	Messages about configuration file processing.
db	Messages that log database operations.
default	Various types of messages. This is the default if nothing is specified.
eventlib	Messages containing debugging data from the event system.
insist	Messages that report internal consistency check failures.
lame–servers	Messages about lame server delegations.
load	Messages about loading the zone.
maintenance	Messages reporting maintenance events.
ncache	Messages about negative caching.
notify	Messages tracing the DNS NOTIFY protocol.
os	Messages reporting operating system problems.
packet	Messages containing dumps of all of the packets sent and received.
panic	Messages generated by a fault that causes the server to shut down.
parser	Messages about configuration command processing.
queries	Messages about every DNS query received.
response–checks	Messages reporting the results of response checking.
security	Messages concerning the application of security criteria. These are most meaningful if allow–update, allow–query, and allow–transfer options are in use.
statistics	Messages containing server statistics.
update	Messages concerning dynamic updates.
xfer–in	Messages recording inbound zone transfers.
xfer–out	Messages recording outbound zone transfers.

The BIND 9 logging statement is very similar to the BIND 8 command. It has the same channel and category clauses, although some of the options in those clauses are different. The BIND 9 logging command syntax is shown in Listing B.4.

Listing B.4: BIND 9 *logging* Command Syntax

```
logging {
  [ channel channel_name {
    ( file pathname
       [ versions number|unlimited ]
       [ size size ]
    |syslog kern|user|mail|daemon|auth|syslog|lpr
                |news|uucp|cron|authpriv|ftp
                |local0|local1|local2|local3
                |local4|local5|local6|local7
    |stderr
    |null;

    [ severity critical|error|warning|notice
                |info|debug [level]|dynamic; ]
    [ print-category yes|no; ]
    [ print-severity yes|no; ]
```

```
    [ print-time yes|no; ]
  }; ]

  [ category category_name {
    channel_name; [ channel_name; ... ]
  }; ]
  ...
};
```

The channel clause is the same as it was in BIND 8, with only the addition of stderr as a possible destination for messages. The category clause looks the same, but the categories have changed. A dozen categories shown in Table B.3 are no longer supported: cname, eventlib, insist, load, maintenance, ncache, os, packet, panic, parser, response–check, and statistics. Ten of the categories listed in Table B.3 remain: config, db, default, lame–server, notify, queries, security, update, xfer–in, and xfer–out, although one category has been renamed from db to database. Six new categories have been added for BIND 9:

general A wide variety of messages.

resolver Messages that relate to DNS resolution.

client Messages that concern processing of client requests.

network Messages that relate to network operations.

dispatch Messages that trace packets sent to various server modules.

dnssec Messages that track the processing of the DNSSEC and TSIG protocols.

Most servers use the default logging configuration, which logs messages through syslogd using the default category. To find out more about custom log configurations, see *Linux DNS Server Administration*, by Craig Hunt, which is also part of the Craig Hunt Linux Library.

The *zone* Statement

The zone statement identifies the zone being served, and defines the source of domain database information. There are four variants of the zone statement: one for the master server, one for the slave servers, one for the hints file, and a special one for forwarding. The syntax of each variant used on BIND 8 systems is shown in Listing B.5.

Listing B.5: BIND 8 *zone* Statement Syntax

```
zone domain_name [ in|hs|hesiod|chaos ] {
  type master;
  file pathname;
  [ forward only|first; ]
  [ forwarders { address-list; }; ]
  [ check-names warn|fail|ignore; ]
  [ allow-update { address_match_list }; ]
  [ allow-query { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ notify yes|no; ]
  [ also-notify { address-list };
  [ dialup yes|no; ]
  [ ixfr-base pathname; ]
```

```
  [ pubkey flags protocol algorithm key; ]
};

zone domain_name [ in|hs|hesiod|chaos ] {
  type slave|stub;
  [ file pathname; ]
  [ ixfr-base pathname; ]
  masters { address-list };
  [ forward only|first; ]
  [ forwarders { address-list; }; ]
  [ check-names warn|fail|ignore; ]
  [ allow-update { address_match_list }; ]
  [ allow-query { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ transfer-source ip_addr; ]
  [ max-transfer-time-in number; ]
  [ notify yes|no; ]
  [ also-notify { address-list };
  [ dialup yes|no; ]
  [ pubkey flags protocol algorithm key; ]
};

zone "." [ in|hs|hesiod|chaos ] {
  type hint;
  file pathname;
  [ check-names warn|fail|ignore; ]
};

zone domain_name [in|hs|hesiod|chaos] {
  type forward;
  [ forward only|first; ]
  [ forwarders { address-list; }; ]
  [ check-names warn|fail|ignore; ]
};
```

The zone statement starts with the keyword zone followed by the name of the domain. For the root cache, the domain name is always ".". The domain name is then followed by the data class. This is always in for Internet DNS service, which is the default if no value is supplied.

The type option defines whether this is a master server, a slave server, or the hints file for the root cache. A stub server is a slave server that loads only the NS records instead of the entire domain.

The file option for a master server points to the source file from which the zone is loaded. For the slave server, it points to the file to which the zone is written. In the root cache statement, the file option points to the hints file used to initialize the cache.

forward, forwarders, check–names, allow–query, allow–transfer, transfer–source, max–transfer–time–in, dialup, notify, and also–notify were all covered in the section on the options statement. Except for the scope of the options, they function the same here. When specified in a zone statement, these options apply only to the specific zone. When specified in the options statement, they apply to all zones. The specific settings for a zone override the global settings of the options statement.

There are a few options that haven't been discussed yet:

> **allow–update** Identifies the hosts that are allowed to dynamically update the zone. By default, no remote system is allowed to modify the zone.

ixfr–base Defines the path to the file where incremental zone file transfers are stored. If you use incremental zone file transfers, upgrade to BIND 9 for a more stable implementation.

pubkey Defines the DNSSEC public encryption key for the zone when there is no trusted mechanism for distributing public keys over the network. pubkey defines the DNSSEC flags, protocol, and algorithm, as well as a base–64 encoded version of the key. The remote server that will be accessing this domain through DNSSEC defines the same settings using the trusted–key command described earlier in this appendix. If you must use encryption for DNS, don't use BIND 8; upgrade to BIND 9.

BIND 9 uses the same four zone command variations as BIND 8. The difference between the two versions of BIND is that BIND 8 and BIND 9 use different options. The BIND 9 syntax of the four zone statement variants is shown in Listing B.6.

Listing B.6: BIND 9 *zone* Statement Syntax

```
zone domain_name [ in|hs|hesiod|chaos ] {
  type master;
  file pathname;
  [ forward only|first; ]
  [ forwarders { address-list; }; ]
  [ allow-update { address_match_list }; ]
  [ allow-update-forwarding { address_match_list }; ]
  [ allow-query { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ allow-notify { address_match_list }; ]
  [ dialup yes|no; ]
  [ notify yes|no|notify|notify-passive|refresh|passive; ]
  [ also-notify { address-list }; ]
  [ database string; [...] ]
  [ update-policy { policy }; ]
  [ sig-validity-interval number; ]
  [ max-refresh-time number; ]
  [ max-retry-time number; ]
  [ max-transfer-idle-out number; ]
  [ max-transfer-time-out number; ]
  [ min-refresh-time number; ]
  [ min-retry-time number; ]
};

zone domain_name [ in|hs|hesiod|chaos ] {
  type slave|stub;
  [ file pathname; ]
  [ ixfr-base pathname; ]
  masters [port ip_port] { address-list };
  [ forward only|first; ]
  [ forwarders { address-list; }; ]
  [ check-names warn|fail|ignore; ]
  [ allow-update { address_match_list }; ]
  [ allow-update-forwarding { address_match_list }; ]
  [ allow-query { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ transfer-source ip_addr; ]
  [ dialup yes|no|notify|notify-passive|refresh|passive; ]
  [ max-transfer-time-in number; ]
  [ notify yes|no; ]
  [ also-notify { address-list }; ]
  [ max-refresh-time number; ]
  [ max-retry-time number; ]
```

```
  [ max-transfer-idle-in number; ]
  [ max-transfer-idle-out number; ]
  [ max-transfer-time-in number; ]
  [ max-transfer-time-out number; ]
  [ min-refresh-time number; ]
  [ min-retry-time number; ]
  [ transfer-source ip_addr|* [port ip_port|*]; ]
  [ transfer-source-v6 ip6_addr|* [port ip_port|*]; ]
};

zone "." [ in|hs|hesiod|chaos ] {
  type hint;
  file pathname;
};

zone domain_name [in|hs|hesiod|chaos] {
  type forward;
  [ forward only|first; ]
  [ forwarders { address-list; }; ]
};
```

Most of the options shown in the BIND 9 syntax were explained in the discussion of the BIND 9 options statement. The two options that are unique to the BIND 9 zone statement are

allow–update–forwarding Identifies the systems that are allowed to submit dynamic zone updates to a slave that will then be forwarded to the master.

database Specifies the type of database used for storing zone data. The default is rbt, which is the only database type supported by the standard BIND 9 executable.

The *server* Statement

The server statement defines the characteristics of a remote server. Listing B.7 is the syntax of the BIND 8 server statement.

Listing B.7: The BIND 8 *server* Statement Syntax

```
server address {
  [ bogus yes|no; ]
  [ support-ixfr yes|no; ]
  [ transfers number; ]
  [ transfer-format one-answer|many-answers; ]
  [ keys { key_id [key_id ... ] }; ]
};
```

transfers and transfer–format were covered in the section on the options statement. Here, they apply to only this remote server. This is particularly useful for the transfer–format command. You can globally set this to the more efficient many–answers format in the options statement and then use individual server statements to fall back to the more compatible one–answer setting for servers that cannot handle the newer format.

The other options included in the server statement are

bogus yes prevents the local server from sending queries to this server. The default is no, which permits queries to the remote server.

support–ixfr yes enables incremental zone file transfers with the remote server. no, the default, disables incremental zone file transfers. Upgrade to BIND 9 if you need incremental zone file transfers.

keys Specifies the key used by the remote host for encryption. This is not implemented in all versions of BIND 8. Use BIND 9 if you need strong authentication.

The BIND 9 server statement syntax varies slightly from the BIND 8 command. It is shown in Listing B.8.

Listing B.8: The BIND 9 *server* Statement Syntax

```
server address {
  [ bogus yes|no; ]
  [ provide-ixfr yes|no; ]
  [ request-ixfr yes|no; ]
  [ transfers number; ]
  [ transfer-format one-answer|many-answers; ]
  [ keys { key_id [key_id ... ] }; ]
};
```

All of the fields are the same as BIND 8, except that support–ixfr has been replaced by two options:

provide–ixfr Indicates that the local server will provide incremental zone transfers to the remote server.

request–ixfr Indicates that the local server will request incremental zone transfers from the remote server.

The *key* Statement

The key statement assigns an internal name, called a key–id, to an algorithm/key pair used for strong authentication. The syntax of the key statement is the same for both BIND 8 and BIND 9. Listing B.9 shows the syntax.

Listing B.9: The *key* Statement Syntax

```
key key_id {
  algorithm algorithm_id;
  secret secret_key;
};
```

key_id The name assigned to the algorithm/key pair.

algorithm_id The authentication algorithm used. At this writing, the only acceptable value for both BIND 8 and BIND 9 is hmac–md5.

secret_key A base–64 encoded key used by the algorithm. Use the dnskeygen utility to generate the *secret_key*.

The *acl* Statement

The acl command assigns a name to an access control list so that it can be referenced elsewhere in the configuration. The syntax of the acl command for both BIND 8 and BIND 9 is shown in Listing B.10.

Listing B.10: The *acl* Statement Syntax

```
acl name {
   access_list
};
```

name An internal name for the list. There are four predefined names:

any Matches every possible address.

none Matches no addresses.

localhost Matches every address assigned to the local host.

localnet Matches every address where the network portion is the same as the network portion of any address assigned to the local host.

access_list A list of IP addresses written in dotted decimal notation with an optional address mask prefix. An exclamation point (!) before an address means "don't match" the value. An *access_list* can also contain the name of a previously defined access control list, including the four predefined names.

The *trusted–keys* Statement

The trusted–keys statement manually defines the public key for a remote domain when that key cannot be securely obtained from the network. Listing B.11 shows the BIND 8 and BIND 9 syntax for the trusted–keys statement.

Listing B.11: The *trusted–keys* Statement Syntax

```
trusted-keys {
   domain_name flags protocol algorithm key; [...]
};
```

domain_name The name of the remote domain.

flags, protocol,* and *algorithm Attributes of the authentication method used by the remote domain. These values are provided by the administrator of the remote domain. They are the flags, protocol, and algorithm field of the remote domain's KEY resource record, which in turn is generated on the remote server using the dnskeygen utility.

key A base−64 encoded string representing the remote domain's public key. This key is obtained from the administrator of the remote domain, who generates it using the dnskeygen utility.

To learn more about how public and private keys are used with DNS, and to learn more about the dnskeygen utility, see *Linux DNS Server Administration* by Craig Hunt, Sybex, 2001.

The *controls* Statement

The BIND 8 controls statement defines the control channels used by ndc. ndc can use a Unix socket or a network socket as a control channel. The controls statement defines those sockets. The BIND 8 controls statement is shown in Listing B.12.

Listing B.12: BIND 8 *controls* Statement Syntax

```
controls {
  [ inet ip_addr
    port ip_port
    allow { address_match_list; }; ]
  [ unix pathname
    perm file_permissions
    owner uid
    group gid; ]
};
```

The first three options (inet, port, and allow) define the IP address and the port number of a network socket and the access control list of those systems allowed to control named through that channel. Because BIND 8 has weak authentication, creating a control channel that is accessible from the network is a risky thing to do. Whoever gains access to that channel has control over the name server process.

The last four options (unix, perm, owner, and group) define the Unix control socket. The Unix socket appears as a file in the filesystem. It is identified by a normal file pathname (for example, /var/run/ndc). Like any file, the Unix socket is assigned the user id (*uid*) of its owner and a valid group id (*gid*). It is protected by standard file permissions. Only numeric *uid*, *gid*, and *file_permissions* values are acceptable. The *file_permissions* value must start with a 0. For example, to set owner read and write, group read, and world no permission, the numeric value would be 0640.

Most BIND 8 configurations do not contain a controls statement because the default configuration does not need to be changed. ndc cannot be used safely over a network, therefore the inet, port, and allow options are not used to configure a network socket. And ndc works locally on the server without any modifications to the UNIX socket. For BIND 8, the default configuration is all that is required.

The BIND 9 controls statement defines the control channels used by rndc. rndc performs the same functions as the older ndc program, but it can reliably be used over a network. The BIND 9 controls statement is shown in Listing B.13.

Listing B.13: BIND 9 *controls* Statement Syntax

```
controls {
  [ inet ip_addr|*
    port ip_port
```

```
    allow address_match_list;
    keys key_list; ]
};
```

The inet, port, and allow options perform the same functions in defining a network socket for BIND 9 as they did for BIND 8, except now they are truly useful because rndc can reliably run over a network socket. To these options, BIND 9 adds a keys option that defines the cryptographic keys used to provide strong authentication for the rndc clients and server.

In BIND 9, the controls statement always defines a network socket. It does not provide options to define a Unix socket. The network socket is always used, even when rndc is run locally from the name server's console. See Chapter 4 for an example of the controls statement that is required to run rndc locally on the server.

BIND 9 *view* Statement

The view statement allows the same zone to be viewed differently by different clients. This makes it possible to provide an internal view to clients within an organization, and a more limited external view to clients in the outside world. The syntax of the view command is shown in Listing B.14.

Listing B.14: The *view* Statement Syntax

```
view view-name {
  match-clients { address_match_list };
  [ view-option; ... ]
  [ zone-statement; ... ]
};
```

view-name An arbitrary name used inside the configuration to identify this view. To prevent conflicts with keywords, *view-name* should be enclosed in quotes, for example, "internal".

match-clients Defines the list of clients that will access the zone through this view.

view-option Any standard BIND 9 option. Any option defined inside the view statement applies only to this view. This allows different options to be applied to the same zone, depending on which view of the zone is being used.

zone-statement A standard BIND 9 zone statement. A complete zone statement is embedded inside the view statement to define the zone accessed through this view.

The view statement is available only in BIND 9. BIND 8 does not support views.

Appendix C: The *m4* Macros for *sendmail*

Overview

This appendix is a quick reference guide to the m4 macros that you might use to construct a sendmail configuration file. (See Chapter 5, "Configuring a Mail Server," for a tutorial on how m4 macros are used.) This appendix describes the syntax and function of the macros; the information is accurate as of the day of publication. For the most current and accurate description of these macros, see the documentation that comes with the sendmail distribution and the README file in the sendmail/cf directory.

Note On Red Hat systems, the sendmail/cf directory is the /usr/share/sendmail–cf directory. To be "distribution–neutral," this appendix refers to the directory as sendmail/cf, but the actual location of the directory varies, depending on the Linux distribution.

Table C.1 lists the sendmail m4 macros, as well as the few built–in m4 commands that commonly appear in sendmail configurations. In this table, and by convention, commands that are part of the m4 language are shown in lowercase, and macros developed specifically for sendmail are shown in uppercase.

Table C.1: The *sendmail m4* Macros

Command	Usage
CANONIFY_DOMAIN	Lists domains that should be converted to canonical name format, even if the nocanonify feature is selected.
CANONIFY_DOMAIN_FILE	Identifies a file that lists domains that should be converted to canonical name format, even if the nocanonify feature is selected.
DAEMON_OPTIONS	Defines runtime options for the sendmail daemon.
define	Defines a value for a configuration variable.
divert	Directs the output of the m4 process.
dnl	Deletes all characters up to and including the next newline character.
DOMAIN	Selects a file containing attributes for your specific domain.
EXPOSED_USER	Lists usernames that should be exempted from masquerading.
FEATURE	Identifies an optional sendmail feature to be included in the configuration.
GENERICS_DOMAIN	Defines fully qualified domains that should be converted by the genericstable database.
GENERICS_DOMAIN_FILE	Identifies a file that lists fully qualified domains that should be converted by the genericstable database.
HACKS	Selects a file containing a locally defined temporary fix.
INPUT_MAIL_FILTER	Defines a mail filter and the variables necessary to call the filter.
LDAPROUTE_DOMAIN	Defines a domain for which mail should be routed based on an LDAP directory entry.
LDAPROUTE_DOMAIN_ FILE	Identifies a file that lists domains for which mail is routed based on entries in an LDAP directory.
LOCAL_CONFIG	

	Marks the start of a section that contains raw sendmail.cf commands.
LOCAL_DOMAIN	Defines an alias hostname for the mail server.
LOCAL_NET_CONFIG	Marks the start of raw rewrite rules that define how mail destined for the local network is handled.
LOCAL_RULE_*n*	Marks the start of a section that contains raw rewrite rules. The *n*, which must be 0, 1, 2, or 3, identifies the ruleset to which the rewrite rules are added.
LOCAL_RULESETS	Marks the start of a ruleset to be added to the configuration.
LOCAL_USER	Lists usernames that should be exempted from relaying even when local mail is being relayed.
MAIL_FILTER	Defines a mail filter.
MAILER	Identifies a set of mailers to be included in the sendmail.cf file.
MAILER_DEFINITIONS	Marks the start of a section containing raw sendmail.cf mailer commands.
MASQUERADE_AS	Defines the domain name used to masquerade outbound mail.
MASQUERADE_DOMAIN	Defines a domain that should be masqueraded.
MASQUERADE_DOMAIN_ FILE	Identifies a file that lists domains that should be masqueraded.
MASQUERADE_ EXCEPTION	Defines a host that should not be masqueraded.
MODIFY_MAILER_FLAGS	Overrides the flags defined for a mailer.
Command	Usage
OSTYPE	Selects a file containing operating system−specific attributes.
RELAY_DOMAIN	Defines a domain for which mail should be relayed.
RELAY_DOMAIN_FILE	Identifies a file that lists domains for which mail should be relayed.
SITE	Identifies a locally connected UUCP host.
SITECONFIG	Identifies the file that lists all of the locally connected UUCP sites.
TRUST_AUTH_MECH	Defines a list of trusted authorization mechanisms.
undefine	Clears the value set for a configuration variable.
UUCPSMTP	Maps a UUCP hostname to an Internet hostname.
VERSIONID	Defines version control information for the configuration.
VIRTUSER_DOMAIN	Defines a virtual domain that will be accepted for processing through the virtusertable.
VIRTUSER_DOMAIN_FILE	Identifies the file that lists virtual domains that will be accepted for processing through the virtusertable.

The four built−in m4 commands, shown in Table C.1 in lowercase characters, are composed of two commands used to control the flow of output and two commands used to set macro values. The two commands that control the flow of output are dnl and divert. The dnl command begins a full− or partial−line comment. The divert(−1) command marks the start of a block of comment text, and the divert(0) command marks the end of the block of comments.

The two built−in m4 commands that set macro values are define and undefine. define sets a variable to a value, and undefine resets it to its default value. More configuration parameters can be controlled through the define command than through any other, so more of this appendix is dedicated to defining variables than to anything else.

Several of the macros—almost half—do essentially the same thing as the define command: They set a variable to a value. MASQUERADE_AS, MASQUERADE_DOMAIN, and VIRTUSER_DOMAIN_FILE are all examples of commands used to set variables. All of these commands are covered later in this appendix.

The TRUST_AUTH_MECH macro is a good example of a macro that complements a define. The define parameter confAUTH_MECHANISMS defines the trust mechanisms advertised to other servers. The TRUST_AUTH_MECH macro is the inverse of this. It identifies the mechanisms accepted from other servers. The same list of keywords is used to configure both confAUTH_MECHANISMS and TRUST_AUTH_MECHANISMS.

The macro names OSTYPE, DOMAIN, FEATURE, MAILER, HACK, and SITECONFIG are all names of subdirectories within the sendmail/cf directory. The value passed to each of these macros is the name of a file within the specified directory. For example, the command OSTYPE(linux) tells m4 to load the file linux.m4 from the ostype directory, and process the m4 source code found there. The .m4 source files pointed to by the OSTYPE, DOMAIN, FEATURE, and MAILER commands are built primarily from define and FEATURE commands.

Two of the macros that are also directory names, SITECONFIG and HACK, are rarely used. The HACK macro points to an m4 source file that contains a temporary site–specific fix to a sendmail problem. You create the file in the hack directory and then use the HACK command to add that file to the configuration. The use of hacks is discouraged and is generally unnecessary.

SITECONFIG points to a source file that contains SITE macros that define the UUCP sites connected to the local host. You create the file containing the SITE macros and then invoke it with the SITECONFIG command. These commands, along with UUCPSMTP, are obsolete and only maintained for backward compatibility.

Despite the fact that older commands such as SITE, SITECONFIG, and UUCPSMTP are obsolete and can be ignored, there are a large number of other commands in Table C.1. Descriptions of these commands make up the rest of this appendix. Use this appendix to help you read existing files and to write your own configuration file.

define

define sets a value used by sendmail. Most "defines" are done in the m4 source files that are called by the .mc file, not in the .mc file itself. Because many define parameters directly affect a single sendmail.cf option, macro, or class, many define statements correspond to individual sendmail.cf command lines. For example, the following define command

```
define(`confMAILER_NAME', `MAILER_DAEMON')
```

placed in an m4 source file has the same effect as the following command

```
DnMAILER_DAEMON
```

placed directly in the sendmail.cf file.

Most of the available configuration parameters are shown in the following sections. Many of these parameters correspond to sendmail options, macros, or classes. In the parameter description list, the name of the corresponding option, macro, or class is shown enclosed in square brackets ([]).

Macro names begin with a dollar sign ($j), class names begin with a dollar sign and an equal sign ($=w), and options are shown with long option names (SingleThreadDelivery).

The list of define parameters is quite long. However, because most of the parameters default to a reasonable value, they do not have to be explicitly set in the m4 source file. The default value of each parameter is shown in the following listing (unless there is no default).

confALIAS_WAIT Sets the amount of time to wait for alias file rebuild. Defaults to 10m. [AliasWait]

confALLOW_BOGUS_HELO Defines special characters that are not normally allowed in DNS hostnames that will be allowed in the hostname on a HELO command line. [AllowBogusHELO]

confAUTH_MECHANISMS Defines a space−separated list of authentication mechan−isms that will be used for SMTP AUTH. [AuthMechanisms] The default is GSSAPI, KERBEROS_V4, DIGEST−MD5, and CRAM−MD5.

confAUTH_OPTIONS The AUTH= argument is added to the MAIL FROM header only when authentication succeeds if this is set to A. [AuthOptions]

confAUTO_REBUILD Automatically rebuilds the alias file if True. Defaults to False. [AutoRebuildAliases]

confBIND_OPTS Sets DNS resolver options. Defaults to undefined. [ResolverOptions]

confBLANK_SUB Defines the character used to replace unquoted blank characters in e−mail addresses. [BlankSub]

confCACERT Identifies a file containing a cryptographic certificate from a certificate authority. [CACERTFile]

confCACERT_PATH Defines the path to the directory that contains the cryptographic certificates. [CACERTPath]

confCF_VERSION Sets the configuration file's version number. [$Z]

confCHECKPOINT_INTERVAL Checkpoints the queue files after this number of queued items are processed. Default is 10. [CheckpointInterval]

confCHECK_ALIASES Looks up every alias during alias file build. Default is False. [CheckAliases]

confCLIENT_CERT Identifies the file containing the cryptographic certificate sendmail uses when it acts as client. [ClientCertFile]

confCLIENT_KEY Identifies the file containing the private key associated with the certificate used when sendmail acts as a client. [ClientKeyFile]

confCLIENT_OPTIONS Defines the port options used for outbound SMTP client connections. [ClientPortOptions]

confCOLON_OK_IN_ADDR Treats colons as regular characters in addresses. Default is False. [ColonOkInAddr]

confCONNECTION_RATE_THROTTLE Sets the maximum number of connections permitted per second. [ConnectionRateThrottle]

confCONNECT_ONLY_TO Defines limited connectivity. Needed only when sendmail is tested by the developers. This is not used in production environments. [ConnectOnlyTo]

confCONTROL_SOCKET_NAME Defines a socket used for managing the sendmail daemon. [ControlSocketName]

confCON_EXPENSIVE Holds mail bound for mailers that have the e flag set until the next queue run. Defaults to False. [HoldExpensive]

confCOPY_ERRORS_TO Sets the address that receives copies of error messages. [PostmasterCopy]

confCR_FILE Points to the file that lists the hosts for which this server will relay mail. Defaults to /etc/mail/relay−domains. [$=R]

confCT_FILE Defines the file of trusted usernames. Defaults to /etc/mail/trusted−users. [$=t]

confCW_FILE Points to the file of local host aliases. Defaults to /etc/mail/local−host−names.cw. [$=w]

confDEAD_LETTER_DROP Defines the file in which failed messages that could not be returned to the sender or sent to the postmaster are saved. [DeadLetterDrop]

confDAEMON_OPTIONS Sets SMTP daemon options. [DaemonPortOptions]

confDEF_AUTH_INFO Identifies the file that contains the authentication information used for outbound connections. [DefaultAuthInfo]

confDEF_CHAR_SET Defines the default character set for unlabeled 8−bit MIME data. Defaults to unknown−8bit. [DefaultCharSet]

confDEF_USER_ID Defines the user ID and group ID used by sendmail. Defaults to 1:1. [DefaultUser]

confDELIVERY_MODE Sets the default delivery mode. Defaults to background. [DeliveryMode]

confDF_BUFFER_SIZE Defines the maximum amount of buffer memory that will be used before a disk file is used. [DataFileBufferSize]

confDH_PARAMETERS Identifies the file that contains the DH parameters for the DSA/DH digital signature algorithm. [DHParameters]

confDIAL_DELAY Sets the time delay before retrying a "dial−on−demand"

connection. Defaults to 0s, which means "don't retry." [DialDelay]

confDOMAIN_NAME Defines the full hostname. [$j]

confDONT_BLAME_SENDMAIL Doesn't perform file security checks if True. Defaults to False. Don't use this option. It is a threat to the security of your server. [DontBlameSendmail]

confDONT_EXPAND_CNAMES Doesn't convert nicknames to canonical names if True. Defaults to False, which means "do convert." [DontExpandCnames]

confDONT_INIT_GROUPS Disables the initgroups(3) routine if True. Defaults to False, which means "use the initgroups(3) routine." [DontInitGroups]

confDONT_PROBE_INTERFACES Doesn't automatically accept the addresses of the server's network interfaces as valid addresses if True. Defaults to False. [DontProbeInterface]

confDONT_PRUNE_ROUTES Doesn't prune route addresses to the minimum possible if True. Defaults to False. [DontPruneRoutes]

confDOUBLE_BOUNCE_ADDRESS When errors occur sending an error message, sends the second error message to this address. Defaults to postmaster. [DoubleBounceAddress]

confEBINDIR Defines the directory in which executables for FEATURE(`local_lmtp') and FEATURE(`smrsh') are stored. The default directory is /usr/libexec.

confEIGHT_BIT_HANDLING Defines how 8−bit data is handled. Defaults to pass8. [EightBitMode]

confERROR_MESSAGE Points to a file containing a message that is prepended to error messages. [ErrorHeader]

confERROR_MODE Defines how errors are handled. Defaults to print. [ErrorMode]

confFALLBACK_MX Defines a backup MX host. [FallbackMXhost]

confFORWARD_PATH Defines places to search for .forward files. Defaults to $z/.forward.$w:$z/.forward. [ForwardPath]

confFROM_HEADER Defines the From: header format. Defaults to $?x$x <$g>$|g. .

confFROM_LINE Defines the format of the Unix From line. Defaults to From $g $d. [UnixFromLine]

confHOSTS_FILE Defines the path to the hosts file. Defaults to /etc/hosts. [HostsFile]

confHOST_STATUS_DIRECTORY Defines the directory in which the host status is saved. [HostStatusDirectory]

confIGNORE_DOTS Ignores dots in incoming messages if True. Defaults to False. [IgnoreDots]

confLDAP_DEFAULT_SPEC Defines the defaults used for LDAP databases unless specifically overridden by a K command for an individual map. [LDAPDefaultSpec]

confLOCAL_MAILER Defines the mailer used for local connections. Defaults to local.

confLOG_LEVEL Defines the level of detail for the log file. Defaults to 9. [LogLevel]

confMAILER_NAME Defines the sender name used on error messages. Defaults to MAILER−DAEMON. [$n]

confMATCH_GECOS Matches the e−mail username to the GECOS field. This match is not done if this parameter is not set. The GECOS field is the "comment" field in the /etc/ passwd file described in Chapter 3, "Login Services." The GECOS field usually contains the user's real name. [MatchGECOS]

confMAX_ALIAS_RECURSION Aliases can refer to other aliases. This sets the maximum depth that alias references can be nested. The default is 10. [MaxAliasRecursion]

confMAX_DAEMON_CHILDREN Refuses connections when this number of children is reached. By default, connections are never refused. [MaxDaemonChildren]

confMAX_HEADERS_LENGTH Defines the maximum length of the sum of all headers in bytes. [MaxHeadersLength]

confMAX_HOP Defines the counter used to determine mail loops. Defaults to 25. [MaxHopCount]

confMAX_MESSAGE_SIZE Sets the maximum size for a message the server will accept. By default, no limit is set. [MaxMessageSize]

confMAX_MIME_HEADER_LENGTH Defines the maximum length of MIME headers. [MaxMimeHeaderLength]

confMAX_QUEUE_RUN_SIZE Defines the maximum number of entries processed in a queue run. Defaults to 0, which means no limit. [MaxQueueRunSize]

confMAX_RCPTS_PER_MESSAGE Defines the maximum number of recipients allowed for a piece of mail. [MaxQueueRunSize]

confMCI_CACHE_SIZE Sets the number of open connections that can be cached. Defaults to 2. [ConnectionCacheSize]

confMCI_CACHE_TIMEOUT Sets the amount of time inactive open connections are held in the cache. Defaults to 5m. [ConnectionCacheTimeout]

confME_TOO Sends a copy to the sender if True. Defaults to False. [MeToo]

confMIME_FORMAT_ERRORS Sends MIME−encapsulated error messages if True. Defaults to True. [SendMimeErrors]

confMIN_FREE_BLOCKS Sets the minimum number of blocks that must be available on the disk to accept mail. Defaults to 100. [MinFreeBlocks]

confMIN_QUEUE_AGE Sets the minimum time a job must be queued. Defaults to 0. [MinQueueAge]

confMUST_QUOTE_CHARS Adds characters to the list of characters that must be quoted when they are included in the user's full name ($x). The characters @,;:\()[] are always quoted. By default, . and ' are added to the list. [MustQuoteChars]

confNO_RCPT_ACTION Defines handling for mail with no recipient headers: do nothing (none); add a To: header (add−to); add an Apparently−To: header (add−apparently−to); add a Bcc: header (add−bcc); add a To: undisclosed−recipients header (add−to−undisclosed). Defaults to none. [NoRecipientAction]

confOLD_STYLE_HEADERS Treats headers without special characters as old style if True. Defaults to True. [OldStyleHeaders]

confOPERATORS Defines the address operator characters. Defaults to .:%@!^/[]+. [OperatorChars]

confPID_FILE Specifies the path of the PID file. [PidFile]

confPRIVACY_FLAGS Sets flags that restrict the use of some mail commands. Defaults to authwarnings. [PrivacyOptions]

confPROCESS_TITLE_PREFIX Identifies the string used on this system as the prefix for the process title in ps listings. [ProcessTitlePrefix]

confQUEUE_FACTOR Defines a value used to calculate when a loaded system should queue mail instead of attempting delivery. Defaults to 600000. [QueueFactor]

confQUEUE_LA Sends mail directly to the queue when this load average is reached. Defaults to 8. [QueueLA]

confQUEUE_SORT_ORDER Sorts queue by Priority or Host order. Defaults to Priority. [QueueSortOrder]

confRECEIVED_HEADER Defines the Received: header format. Defaults to $?sfrom $s $.$?_($?s$|from $.$_) $.by $j ($v/$Z)$?r with r. id i?u for u.; $b.

confRAND_FILE Identifies the file that contains random data needed by STARTTLS if sendmail was not compiled with the HASURANDOM flag. [RandFile]

confRECEIVED_HEADER Defines the Received: header format. Defaults to $?sfrom $s $.$?_($?s$|from $.$_) $.by $j ($v/$Z)$?r with r. id i?u for u.; $b .

confREFUSE_LA Defines the load average at which incoming connections are refused. Defaults to 12. [RefuseLA]

confREJECT_MSG Defines the message displayed when mail is rejected because of the access control database. Defaults to 550 Access denied.

confRELAY_MAILER Defines the default mailer name for relaying. Defaults to relay.

confRRT_IMPLIES_DSN True tells sendmail to interpret a Return−Receipt−To: header as a request for delivery status notification (DSN). The default is false. [RrtImpliesDsn]

confRUN_AS_USER Runs as this user to read and deliver mail. By default, this is not used. [RunAsUser]

confSAFE_FILE_ENV Chroot() to this directory before writing files. By default, this is not done. [SafeFileEnvironment]

confSAFE_QUEUE Creates a queue file; then attempts delivery. This is not done unless this parameter is specified. [SuperSafe]

confSAVE_FROM_LINES Does not discard Unix From lines. They are discarded if this is not set. [SaveFromLine]

confSEPARATE_PROC Delivers messages with separate processes if True. Defaults to False. [ForkEachJob]

confSERVER_CERT Identifies the file that contains the cryptographic certificate used when this system acts as a server. [ServerCertFile]

confSERVER_KEY Identifies the file that contains the private key associated with the cryptographic certificate used when this system acts as a server. [ServerKeyFile]

confSERVICE_SWITCH_FILE Defines the path to the service switch file. Defaults to / etc/service.switch. [ServiceSwitchFile]

confSEVEN_BIT_INPUT Forces input to seven bits if True. Defaults to False. [SevenBitInput]

confSINGLE_LINE_FROM_HEADER Forces a multiline From: line to a single line when True. Defaults to False. [SingleLineFromHeader]

confSINGLE_THREAD_DELIVERY Forces single−threaded mail delivery when True and HostStatusDirectory is defined. Defaults to False. [SingleThreadDelivery]

confSMTP_LOGIN_MSG Defines the SMTP greeting message. Defaults to $j Sendmail $v/$Z; $b. [SmtpGreetingMessage]

confSMTP_MAILER Defines the mailer used for SMTP connections; must be smtp, smtp8, or esmtp. Defaults to esmtp.

confTEMP_FILE_MODE Sets the file mode used for temporary files. Defaults to 0600. [TempFileMode]

confTIME_ZONE Sets the time zone from the system (USE_SYSTEM) or the TZ variable (USE_TZ). Defaults to USE_SYSTEM. [TimeZoneSpec]

confTO_COMMAND Sets the maximum time to wait for a command. Defaults to 1h. [Timeout.command]

confTO_CONNECT Sets the maximum time to wait for a connect. [Timeout.connect]

confTO_CONTROL Sets the maximum amount of time allowed for a control socket transaction to complete. The default is two minutes (2m). [Timeout.control]

confTO_DATABLOCK Sets the maximum time to wait for a block during DATA phase. Defaults to 1h. [Timeout.datablock]

confTO_DATAFINAL Sets the maximum time to wait for a response to the terminating ".". Defaults to 1h. [Timeout.datafinal]

confTO_DATAINIT Sets the maximum time to wait for a DATA command response. Defaults to 5m. [Timeout.datainit]

confTO_FILEOPEN Sets the maximum time to wait for a file open. Defaults to 60s. [Timeout.fileopen]

confTO_HELO Sets the maximum time to wait for a HELO or EHLO response. Defaults to 5m. [Timeout.helo]

confTO_HOSTSTATUS Sets the timer for stale host status information. Defaults to 30m. [Timeout.hoststatus]

confTO_ICONNECT Sets the maximum time to wait for the very first connect attempt to a host. [Timeout.iconnect]

confTO_IDENT Sets the maximum time to wait for an IDENT query response. Defaults to 30s. [Timeout.ident]

confTO_INITIAL Sets the maximum time to wait for the initial connect response. Defaults to 5m. [Timeout.initial]

confTO_MAIL Sets the maximum time to wait for a MAIL command response. Defaults to 10m. [Timeout.mail]

confTO_MISC Sets the maximum time to wait for other SMTP command responses. Defaults to 2m. [Timeout.misc]

confTO_QUEUERETURN_NONURGENT Sets the "Undeliverable mail" timeout for low–priority messages. [Timeout.queuereturn.non–urgent]

confTO_QUEUERETURN_NORMAL Sets the "Undeliverable mail" timeout for normal–priority messages. [Timeout.queuereturn.normal]

confTO_QUEUERETURN_URGENT Sets the "Undeliverable mail" timeout for

urgent–priority messages. [Timeout.queuereturn.urgent]

confTO_QUEUERETURN Sets the time until a message is returned from the queue as undeliverable. Defaults to 5d. [Timeout.queuereturn]

confTO_QUEUEWARN_NONURGENT Sets the time until a "still queued" warning is sent for low–priority messages. [Timeout.queuewarn.non–urgent]

confTO_QUEUEWARN_NORMAL Sets the time until a "still queued" warning is sent for normal priority messages. [Timeout.queuewarn.normal]

confTO_QUEUEWARN_URGENT Sets the time until a "still queued" warning is sent for urgent priority messages. [Timeout.queuewarn.urgent]

confTO_QUEUEWARN Sets the time until a "still queued" warning is sent about a message. Defaults to 4h. [Timeout.queuewarn]

confTO_QUIT Sets the maximum time to wait for a QUIT command response. Defaults to 2m. [Timeout.quit]

confTO_RCPT Sets the maximum time to wait for a RCPT command response. Defaults to 1h. [Timeout.rcpt]

confTO_RESOLVER_RETRANS Defines, in seconds, the retransmission timer for all resolver lookups. [Timeout.resolver.retrans]

confTO_RESOLVER_RETRANS_FIRST Defines, in seconds, the retransmission timer for the resolver lookup for the first attempt to deliver a message. [Timeout.resolver.retrans.first]

confTO_RESOLVER_RETRANS_NORMAL Defines, in seconds, the retransmission timer for all resolver lookups after the first attempt to deliver a message. [Timeout.resolver.retrans.normal]

confTO_RESOLVER_RETRY Defines the total number of times to retry a resolver query. [Timeout.resolver.retry]

confTO_RESOLVER_RETRY_FIRST Defines the number of times the resolver query for the first delivery attempt is retried. [Timeout.resolver.retry.first]

confTO_RESOLVER_RETRY_NORMAL Defines the number of times to retry resolver queries after the first delivery attempt. [Timeout.resolver.retry.normal]

confTO_RSET Sets the maximum time to wait for a RSET command response. Defaults to 5m. [Timeout.rset]

confTRUSTED_USER Defines the user who controls the sendmail daemon, and owns the files created by sendmail. Do not confuse this option with confTRUSTED_USERS. [TrustedUser]

confTRUSTED_USERS Defines trusted usernames to add to root, uucp, and daemon.

confTRY_NULL_MX_LIST Connects to the remote host directly if the MX points to the local host and it is set to True. Defaults to False. [TryNullMXList]

confUNSAFE_GROUP_WRITES Doesn't reference programs or files from group-writable :include: and .forward files if True. Defaults to False. [UnsafeGroupWrites]

confUSERDB_SPEC Defines the path of the user database file. [UserDatabaseSpec]

confUSE_ERRORS_TO Delivers errors using the Errors-To: header if True. Defaults to False. [UserErrorsTo]

confUUCP_MAILER Defines the default UUCP mailer. Defaults to uucp-old.

confWORK_CLASS_FACTOR Defines the factor used to favor high-priority jobs. Defaults to 1800. [ClassFactor]

confWORK_RECIPIENT_FACTOR Defines the factor used to lower the priority of a job for each additional recipient. Defaults to 30000. [RecipientFactor]

confWORK_TIME_FACTOR Defines the factor used to lower the priority of a job for each delivery attempt. Defaults to 90000. [RetryFactor]

define macros are the most common macros in the m4 source files. The next most commonly used macro is the FEATURE macro.

FEATURE

The FEATURE macro processes m4 source code from the feature directory. Source files in that directory define optional sendmail features. The syntax of the FEATURE macro is

FEATURE(*name*, [*argument*])

The *argument* is optional. If an argument is passed to the source file, it is used by the source file to generate code for the sendmail.cf file. For example, the following generates the code for accessing the mailertable, and defines that table as being a dbm database located in the file /usr/lib/mailertable:

```
FEATURE(mailertable, dbm /usr/lib/mailertable)
```

The available features and their purposes are listed in Table C.2.

Table C.2: Optional sendmail Features

Name	Purpose
accept_unqualified_senders	Allows network mail from addresses that do not include a valid hostname.
accept_unresolvable_domains	Accepts mail from hosts that are unknown to DNS.
access_db	Enables the use of the access database.

allmasquerade	Also masquerades recipient addresses.
always_add_domain	Adds the local hostname to all locally delivered mail.
bestmx_is_local	Accepts mail addressed to a host that lists the local system as its MX server as local.
bitdomain	Uses a table to map Bitnet hosts to Internet addresses.
blacklist_recipients	Filters incoming mail based on values set in the access database.
delay_checks	Delay the check_mail and check_relay rulesets until check_rcpt is called.
dnsbl	Reject mail from hosts listed in a DNS−based rejection list. Replaces rbl.
domaintable	Uses a domain table for domain name mapping.
generics_entire_domain	Map domain names identified in class G through the genericstable.
genericstable	Uses a table to rewrite local addresses.
ldap_routing	Enable LDAP−based e−mail routing.
limited_masquerade	Only masquerade hosts listed in $=M.
local_lmtp	Uses mail.local with LMTP support.
local_procmail	Uses procmail as the local mailer.
loose_relay_check	Disables validity checks for addresses that use the % hack.
mailertable	Routes mail using a mailer table.
masquerade_entire_domain	Masquerades all hosts within the masquerading domains.
masquerade_envelope	Masquerades the envelope sender address in addition to the header sender address.
no_default_msa	Allows the default configuration of the Message Submission Agent to be overridden by the DAEMON_OPTIONS macro.
nocanonify	Doesn't convert names with $[... $] syntax.
nodns	Doesn't include DNS support.
nouucp	Doesn't include UUCP address processing.
nullclient	Forwards all mail to a central server.
Name	Purpose
promiscuous_relay	Relays mail from any site to any site.
rbl	Enables use of the Realtime Blackhole List server. Replaced by dnsbl.
redirect	Supports the .REDIRECT pseudo−domain.
relay_based_on_MX	Relays mail for any site whose MX points to this server.
relay_entire_domain	Relays mail for any host in your domain.
relay_host_only	Relays mail only for hosts listed in the access database.
relay_local_from	Relays mail if the source is a local host.
relay_mail_from	Relays mail if the sender is listed as RELAY in the access database.
smrsh	Uses smrsh as the prog mailer.

stickyhost	Treats user differently from *user@local.host*.
use_ct_file	Loads $=t from the file defined by confCT_FILE.
use_cw_file	Loads $=w from the file defined by confCW_FILE.
uucpdomain	Uses a table to map UUCP hosts to Internet addresses.
virtuser_entire_domain	Maps entire domain names through the virtusertable.
virtusertable	Maps virtual domain names to real mail addresses.

The use_cw_file and the use_ct_file features are equivalent to Fw/etc/mail/local–host–names and Ft/etc/mail/trusted–users commands in the sendmail.cf file. See Chapter 5 for descriptions of host aliases ($=w) and trusted users ($=t). The redirect feature is also covered in Chapter 5.

Several FEATURE macros remove unneeded lines from the sendmail.cf file. nouucp removes the code that handles UUCP addresses for systems that do not have access to UUCP networks, and nodns removes the code for DNS lookups for systems that do not have access to DNS, or do not want to use DNS. nocanonify disables the code that converts nicknames and IP addresses into hostnames. Finally, the nullclient feature strips everything out of the configuration, except for the capability to forward mail to a single mail server via a local SMTP link. The name of that mail server is provided as the argument on the nullclient command line, for example, FEATURE(nullclient, big.isp.net) forwards all mail to big.isp.net without any local mail processing.

Several features relate to mail relaying and masquerading. They are stickyhost, allmasquerade, limited_masquerade, and masquerade_entire_domain. All of these features are covered in the DOMAIN section later in this appendix.

Several of the features define databases that are used to perform special address processing. All of these features accept an optional argument that defines the database. (See the sample mailertable command at the beginning of this section for an example of defining the database with the optional argument.) If the optional argument is not provided, the database description always defaults to hash –o /etc/*filename*, where *filename* matches the name of the feature. For example, mailertable defaults to the definition hash –o /etc/mailertable. The database features are as follows:

access_db Controls mail–relaying and delivery. The access file contains two fields: an e–mail address, which is the key, and an action taken for mail containing that address. The access database is covered in Chapter 11, "More Mail Services."

mailertable Maps host and domain names to specific mailer:host pairs. The mailer, host, user triple is returned by ruleset parse based on the delivery address. The mailertable file allows you to define the mailer and the host of the delivery triple based on the domain name in the delivery address. If the host or domain name in the delivery addresses matches a key field in the mailertable database, it returns the mailer and host for that address. The format of a mailertable entry is

```
domain-name      mailer:host
```

where *domain–name* is either a full hostname (host plus domain) or a domain name. If a domain name is used, it must start with a dot (.), and it will match every host in the specified domain. *mailer* is the internal sendmail.cf mailer name of the mailer that handles mail for the specified domain, and *host* is the hostname of the mailer server that handles mail for that domain.

domaintable Converts an old domain name to a new domain name. The old name

is the key, and the new name is the value returned for the key.

bitdomain Converts a Bitnet hostname to an Internet hostname. The Bitnet name is the key, and the Internet hostname is the value returned. The bitdomain program that comes with the sendmail distribution can be used to build this database. Bitnet is obsolete.

uucpdomain Converts a UUCP name to an Internet hostname. The key is the UUCP hostname, and the value returned is the Internet hostname. This is useful only if you still have users who address e−mail using old UUCP addresses.

genericstable Converts a sender e−mail address. The key to the database is either a username or a full e−mail address (username and hostname). The value returned by the database is the new e−mail address. (See Chapter 5 for an example of using the genericstable.) If you use the genericstable and you don't use masquerading, use generics_domain and generics_ domain_file to get the same functions normally provided by masquerade_domain and masquerade_domain_file.

virtusertable Aliases incoming e−mail addresses. Essentially, this is an extended alias database for aliasing addresses that are not local to this host. The key to the database is a full e−mail address or a domain name. The value returned by the database is the recipient address to which the mail is delivered. If a domain name is used as a key, it must begin with an at sign (@). Mail addressed to any user in the specified domain is sent to the recipient defined by the virtusertable database. Any hostname used as a key in the virtusertable database must also be defined in class w.

Some features are important in the fight against spam because they help control the mail a server delivers or forwards on for delivery. These are accept_unqualified_senders, accept_unresolvable_domains, access_db, blacklist_recipients, and dnsbl. All of these are covered in the section on controlling spam in Chapter 11.

Two of the remaining FEATURE commands relate to domains. The always_add_domain macro makes sendmail add the local domain name to all locally delivered mail, even to those pieces of mail that would normally have just a username as an address. The bestmx_is_local feature accepts mail addressed to a host that lists the local host as its preferred MX server as if the mail were local mail. If this feature is not used, mail bound for a remote host is sent directly to the remote host even if its MX record lists the local host as its preferred MX server. The bestmx_is_local feature should not be used if you use a wildcard MX record for your domain.

The last two features are used to select optional programs for the local and the prog mailers. local_ procmail selects procmail as the local mailer. Provide the path to procmail as the argument in the FEATURE command. The smrsh feature selects the SendMail Restricted SHell (smrsh) as the prog mailer. smrsh provides improved security over /bin/sh, which is often used as the prog mailer. Provide the path to smrsh as the argument in the FEATURE command.

The FEATURE commands discussed in this section and the define macros discussed previously are used to build the m4 source files. The next few sections of this appendix describe the purpose and structure of the OSTYPE, DOMAIN, and MAILER source files.

OSTYPE

OSTYPE points to the m4 source file that contains the operating system specific information for this configuration. This required file is examined in detail in Chapter 5.

Although all m4 macros can be used in OSTYPE source files, Table C.3 lists the define parameters most frequently associated with the OSTYPE file and the function of each parameter. If the parameter has a default value, it is shown enclosed in square brackets after the parameter's functional description.

Table C.3: OSTYPE defines

Parameter	Function
ALIAS_FILE	Name of the alias file. [/etc/aliases]
CYRUS_BB_MAILER_ARGS	cyrusbb mailer arguments. [deliver –e –m $u]
CYRUS_BB_MAILER_FLAGS	Flags added to lsDFMnP for the cyrusbb mailer.
CYRUS_MAILER_ARGS	cyrus mailer arguments. [deliver –e –m $h –– $u]
CYRUS_MAILER_FLAGS	Flags added to lsDFMnP for the cyrus mailer. [A5@]
CYRUS_MAILER_MAX	Maximum size message for the cyrus mailer.
CYRUS_MAILER_PATH	Path to the cyrus mailer. [/usr/cyrus/bin/ deliver]
CYRUS_MAILER_USER	User and group used to the cyrus mailer. [cyrus:mail]
DSMTP_MAILER_ARGS	dsmtp mailer arguments. [IPC $h]
ESMTP_MAILER_ARGS	esmtp mailer arguments. [IPC $h]
FAX_MAILER_ARGS	FAX mailer arguments. [mailfax $u $h $f]
FAX_MAILER_MAX	Maximum size of a FAX. [100000]
FAX_MAILER_PATH	Path to the FAX program. [/usr/local/lib/fax/ mailfax]
HELP_FILE	Name of the help file. [/usr/lib/sendmail.hf]
LOCAL_MAILER_ARGS	Arguments for local mail delivery. [mail –d $u]
LOCAL_MAILER_CHARSET	Character set for local 8–bit MIME mail.
LOCAL_MAILER_DSN_DIAGNOSTIC_ CODE	The delivery status notification code used for local mail. [X–Unix]
LOCAL_MAILER_EOL	The end–of–line character for local mail.
LOCAL_MAILER_FLAGS	Local mailer flags added to lsDFM. [rmn]
LOCAL_MAILER_MAX	Maximum size of local mail.
LOCAL_MAILER_MAXMSG	The maximum number of messages delivered with a single connection.
LOCAL_MAILER_PATH	The local mail delivery program. [/bin/mail]
LOCAL_SHELL_ARGS	Arguments for the prog mail. [sh –c $u]
LOCAL_SHELL_DIR	Directory that the shell should run. [$z:/]
LOCAL_SHELL_FLAGS	Flags added to lsDFM for the shell mailer. [eu]

LOCAL_SHELL_PATH	Shell used to deliver piped e−mail. [/bin/sh]
MAIL11_MAILER_ARGS	mail11 mailer arguments. [mail11 $g $x $h $u]
MAIL11_MAILER_FLAGS	Flags for the mail11 mailer. [nsFx]
Parameter	Function
MAIL11_MAILER_PATH	Path to the mail11 mailer. [/usr/etc/mail11]
PH_MAILER_ARGS	phquery mailer arguments. [phquery −− $u]
PH_MAILER_FLAGS	Flags for the phquery mailer. [ehmu]
PH_MAILER_PATH	Path to the phquery program. [/usr/local/etc/phquery]
POP_MAILER_ARGS	POP mailer arguments. [pop $u]
POP_MAILER_FLAGS	Flags added to lsDFM for the POP mailer. [Penu]
POP_MAILER_PATH	Path of the POP mailer. [/usr/lib/mh/spop]
PROCMAIL_MAILER_ARGS	procmail mailer arguments. [procmail −m $h $f $u]
PROCMAIL_MAILER_FLAGS	Flags added to DFMmn for the procmail mailer. [Shu]
PROCMAIL_MAILER_MAX	Maximum size message for the procmail mailer.
PROCMAIL_MAILER_PATH	Path to the procmail program. [/usr/local/bin/procmail]
QPAGE_MAILER_ARGS	qpage mailer arguments. [qpage −10 −m −P$u]
QPAGE_MAILER_FLAGS	Flags for the qpage mailer. [mDFMs]
QPAGE_MAILER_MAX	Maximum qpage mailer message size. [4096]
QPAGE_MAILER_PATH	Path of the qpage mailer. [/usr/local/bin/qpage]
QUEUE_DIR	Directory containing queue files. [/var/spool/mqueue]
RELAY_MAILER_ARGS	relay mailer arguments. [IPC $h]
RELAY_MAILER_FLAGS	Flags added to mDFMuX for the relay mailer.
RELAY_MAIL_MAXMSG	The maximum number of messages for the relay mailer delivered by a single connection.
SMTP8_MAILER_ARGS	smtp8 mailer arguments. [IPC $h]
SMTP_MAILER_ARGS	smtp mailer arguments. [IPC $h]
SMTP_MAILER_CHARSET	Character set for SMTP 8−bit MIME mail.
SMTP_MAILER_FLAGS	Flags added to mDFMUX for all smtp mailers.
SMTP_MAILER_MAX	Maximum size of messages for all smtp mailers.
SMTP_MAIL_MAXMSG	The maximum number of smtp messages delivered by a single connection.
STATUS_FILE	Name of the status file. [/etc/sendmail.st]
USENET_MAILER_ARGS	Arguments for the usenet mailer. [−m −h −n]
USENET_MAILER_FLAGS	usenet mailer flags. [rlsDFMmn]
USENET_MAILER_MAX	Maximum size of usenet mail messages. [100000]
USENET_MAILER_PATH	Program used for news. [/usr/lib/news/inews]
UUCP_MAILER_ARGS	

	UUCP mailer arguments. [uux – –r –z –a$g –gC $h!rmail ($u)]
UUCP_MAILER_CHARSET	Character set for UUCP 8–bit MIME mail.
UUCP_MAILER_FLAGS	Flags added to DFMhuU for the UUCP mailer.
UUCP_MAILER_MAX	Maximum size for UUCP messages. [100000]
UUCP_MAILER_PATH	Path to the UUCP mail program. [/usr/bin/uux]

DOMAIN

The DOMAIN macro identifies the m4 source file that contains configuration information specific to the local domain. Chapter 5 provides a detailed example of creating a domain source file and then calling that file with the DOMAIN macro.

Table C.4 lists the define macros that commonly appear in DOMAIN source files. All of these define mail relay hosts. The value provided for each parameter is either a hostname (that is, the name of a mail relay server); or a *mailer*:*hostname* pair, where *mailer* is an internal mailer name and *hostname* is the name of the mail relay server. If only a hostname is used, the mailer defaults to relay, which is the name of the SMTP relay mailer.

Table C.4: Mail Relay defines

Parameter	Function
UUCP_RELAY	Server for UUCP–addressed e–mail.
BITNET_RELAY	Server for BITNET–addressed e–mail.
DECNET_RELAY	Server for DECNET–addressed e–mail.
FAX_RELAY	Server for mail to the .FAX pseudo–domain. The fax mailer overrides this value.
LOCAL_RELAY	Server for unqualified names. This is obsolete.
LUSER_RELAY	Server for local names that really aren't local.
MAIL_HUB	Server for all incoming mail.
SMART_HOST	Server for all outgoing mail.

The precedence of the relays defined by these parameters is from the most specific to the least specific. If both the UUCP_RELAY and the SMART_HOST relay are defined, the UUCP_ RELAY is used for outgoing UUCP mail, even though the SMART_HOST relay is defined as handling "all" outgoing mail. If you define both LOCAL_RELAY and MAIL_HUB, use the FEATURE(stickyhost) command. When the stickyhost feature is specified, LOCAL_RELAY handles all local addresses that do not have a host part, and MAIL_HUB handles all local addresses that do have a host part. If stickyhost is not specified, and both relays are defined, the LOCAL_RELAY is ignored, and MAIL_HUB handles all local addresses.

In addition to the defines shown in Table C.3, macros that relate to masquerading and relaying also appear in the DOMAIN source file. The macros are as follows:

> **EXPOSED_USER(*username*)** Disables masquerading when the user portion of the sender address matches *username*. Some usernames, such as root, occur on many systems, and therefore are not unique across a domain. For those usernames, converting the host portion of the address makes it impossible to sort out where the

message really came from, and makes replies impossible. This command prevents the MASQUERADE_AS macro from having an effect on the sender addresses for specific users. This is the same as setting the values in class E in the sendmail.cf file.

LOCAL_USER(*usernames*) Defines local usernames that should not be relayed, even if LOCAL_RELAY or MAIL_HUB are defined. This command is the same as adding usernames to class L in the sendmail.cf file.

MASQUERADE_AS(*host.domain*) Converts the host portion of the sender address on outgoing mail to the specified domain name. Sender addresses that have no hostname or that have a hostname found in the w class are converted. This has the same effect as the M macro in the sendmail.cf file. See examples of MASQUERADE_AS and macro M in Chapter 5.

MASQUERADE_DOMAIN(*otherhost.domain*) Converts the host portion of the sender address on outgoing mail to the domain name defined by the MASQUERADE_AS command if the host portion of the sender address matches the value defined here. This command must be used in conjunction with MASQUERADE_AS. Its effect is the same as adding hostnames to class M in the sendmail.cf file. See Chapter 5.

MASQUERADE_DOMAIN_FILE(*filename*) Loads class M hostnames from the specified file. This can be used in place of multiple MASQUERADE_DOMAIN commands. Its effect is the same as using the FM*filename* command in the sendmail.cf file.

MASQUERADE_EXCEPTION(*host.domain*) This macro defines a host that is not masqueraded, even if it belongs to a domain that is being masqueraded. This allows you to masquerade an entire domain with the MASQUERADE_DOMAIN macro and then exempt a few hosts that should be exposed to the outside world.

RELAY_DOMAIN(*otherhost.domain*) This macro identifies a host for which mail should be relayed. The host identified in this manner is added to class R.

RELAY_DOMAIN_FILE(*filename*) This macro identifies a file that contains a list of hosts for which mail should be relayed. This macro loads class R from the specified file.

There are also several features that affect relaying and masquerading. One, FEATURE (stickyhost), was already discussed. Others are the following:

FEATURE(masquerade_envelope) Causes envelope addresses to be masqueraded in the same way that sender addresses are masqueraded. See Chapter 5 for an example of this command.

FEATURE(allmasquerade) Causes recipient addresses to be masqueraded in the same way that sender addresses are masqueraded. Thus, if the host portion of the recipient address matches the requirements of the MASQUERADE_AS command, it is converted. Don't use this feature unless you are positive that every alias known to the local system is also known to the mail server that handles mail for the masquerade domain.

FEATURE(limited_masquerade) Limits masquerading to those hosts defined in class M. The hosts defined in class w are not masqueraded.

FEATURE(masquerade_entire_domain) Causes MASQUERADE_DOMAIN to be interpreted as referring to all hosts within an entire domain. If this feature is not used, only an address that exactly matches the value defined by MASQUERADE_DOMAIN is converted. If this feature is used, then all addresses that end with the value defined by MASQUERADE_DOMAIN are converted. For example, assume that MASQUERADE_AS(foobirds.org) and MASQUERADE_DOMAIN(swans.foobirds.org) are defined. If FEATURE(masquerade_ entire_domain) is set, every hostname in the swans.foobirds.org domain is converted to foobirds.org on outgoing e–mail. Otherwise, only a host named swans.foobirds.org is converted.

Some features define how the server handles mail if it is the mail relay server. These features, which are also described in Chapter 11, are the following:

access_db Maps a user, a domain name, or an IP address to a keyword that tells sendmail how to handle relaying for the host, domain, or network. This database is used in Chapter 11.

blacklist_recipient Uses the access database to control delivery of mail based on the recipient address. The basic access_db feature controls relaying and delivery based on the source of the message. This feature adds to the capability to control mail relaying and delivery based on the destination.

dnsbl Controls mail delivery based on a DNS blacklist. Source addresses and destination addresses listed in the DNS database may be denied mail delivery or relay services.

promiscuous_relay Relays from any site to any site. Normally, sendmail does not relay mail. Using this feature is a bad idea because it makes you a possible relay server for spammers.

relay_entire_domain Relays from any domain defined in class M to any site.

relay_hosts_only Relays mail from any host defined in the access database or class R.

relay_based_on_MX Relays mail from any site for which your system is the MX server.

relay_local_from Relays mail with a sender address that contains your local domain name.

The DOMAIN source file is also used for features and macros that directly relate to DNS. These features and macros include the following:

FEATURE(accept_unqualified_senders) Accepts mail from the network even if the sender address does not include a hostname. Normally, only mail from a user directly logged on to the system is accepted without a hostname. This is a dangerous feature that should be used only on an isolated network.

FEATURE(accept_unresolvable_domains) Accepts mail from hostnames that cannot be resolved by DNS. This is a dangerous feature that is used only on systems that lack full–time DNS service, such as mobile laptops.

FEATURE(always_add_domain) Adds the hostname of the system to all local mail. With this feature enabled on a server named ibis.foobirds.org, mail from the local user craig to the local user kathy would be delivered as mail from craig@ibis.foobirds.org to kathy@ibis.foobirds.org.

FEATURE(bestmx_is_local) Accepts mail addressed to any host that lists the sendmail server as its MX server as local mail.

CANONIFY_DOMAIN(*domain*) Defines a domain name that will be passed to DNS for conversion to its canonical form, even if the nocanonify feature is in use. This macro is generally used to enable canonification of the local domain when nocanonify is in use.

CANONIFY_DOMAIN_FILE(*filename*) Identifies a file containing a list of domain names that should be converted to canonical form, even if nocanonify has been selected.

LOCAL_DOMAIN(*alias*–hostname) Defines an alias for the local host. Mail addressed to the alias will be accepted as if it were addressed directly to the local host.

The macros and features described in this section are not limited to the DOMAIN source file. They can appear in any m4 source file, and, in fact, are often found in the macro control file. They are listed here because they are most naturally associated with the DOMAIN file.

MAILER

The MAILER command identifies an m4 source file that contains the configuration commands that define a sendmail mailer. A least one MAILER command must appear in the configuration file. Generally more than one MAILER command is used.

It is possible that you will need to customize a file location in an OSTYPE file, or that you will need to define domain–specific information in a DOMAIN file. Unless you develop your own mail–delivery program, however, you will not need to create a MAILER source file. Instead, you will need to invoke one or more existing files in your macro configur–ation file.

Table C.5 lists each MAILER name and its function. These are invoked using the MAILER(*name*) command in the macro configuration (.mc) file.

Table C.5: MAILER Values

Name	Function
local	The local and prog mailers.
smtp	All SMTP mailers: smtp, esmtp, smtp8, dsmtp, and relay.
uucp	All UUCP mailers: uucp–old (uucp) and uucp–new (suucp).
usenet	Usenet news support.

fax	FAX support using FlexFAX software.
pop	Post Office Protocol (POP) support.
procmail	An interface for procmail.
mail11	The DECnet mail11 mailer.
phquery	The phquery program for CSO phone book.
qpage	The QuickPage mailer used to send e−mail to a pager.
cyrus	The cyrus and cyrusbb mailers.

Your macro configuration file should have a MAILER(local) and a MAILER(smtp) entry. Selecting local and smtp provides everything you need for a standard TCP/IP installation. None of the remaining mailers is widely used. The other mailers are the following:

uucp Provides UUCP mail support for systems directly connected to UUCP networks. The uucp−old mailer supports standard UUCP mail, and the uucp−new mailer is used for remote sites that can handle multiple recipients in one transfer. Specify MAILER(uucp) after the MAILER(smtp) entry if your system has both TCP/IP and UUCP connections.

usenet Sends local mail that contains .usenet in the recipient name to the program inews. Use a user mail agent that supports Usenet news. Don't hack sendmail to handle it.

fax Experimental support for HylaFAX.

pop On Linux systems, POP support is provided by the popd, so the MAILER(pop) command is not used.

procmail Provides a procmail interface for the mailertable.

mail11 Used only on DECNET mail networks that use the mail11 mailer.

phquery Provides CSO phone book (ph) directory service.

qpage This mailer provides an interface from e−mail to pagers using the QuickPage program.

cyrus Provides a local mail delivery program that uses a mailbox architecture. cyrus and cyrusbb mailers are not widely used.

Local Code

There are several m4 macros that allow you to directly modify the sendmail.cf file with unadulterated sendmail.cf configuration commands. These macros are placed at the beginning of a block of sendmail.cf code, and they tell m4 where to put that code in the output file. These macros are as follows:

LOCAL_RULE LOCAL_RULE_*n* heads a section of code to be added to ruleset *n*, where *n* is 0, 1, 2, or 3. The code that follows the LOCAL_RULE command is sendmail.cf rewrite rules.

LOCAL_CONFIG LOCAL_CONFIG heads a section of code to be added to the sendmail.cf file after the local information section and before the rewrite rules. The section of code contains standard sendmail.cf configuration commands.

LOCAL_RULESETS This macro heads a section of code that contains a complete ruleset that is to be added to the sendmail.cf file. Generally, these are named as opposed to numbered rulesets.

LOCAL_NET_CONFIG This macro heads a section of sendmail.cf rewrite rules that defines how mail addressed to systems on the local network is handled.

MAILER_DEFINITIONS This macro is placed before a sendmail.cf M command, which is a mailer definition.

DAEMON_OPTIONS

The DAEMON_OPTIONS macro defines parameters for the sendmail daemon. When sendmail accepts mail from a local e–mail program, it is acting as a Mail Submission Agent (MSA). When it transfers that mail to a remote server, it is acting as a Mail Transfer Agent (MTA). The DAEMON_OPTIONS macro sets options for both of sendmail's "personalities."

Two DAEMON_OPTIONS commands are needed to set the parameters for both the MTA and the MSA. The sendmail configuration defaults to the following values:

```
DAEMON_OPTIONS(`Port=25, Name=MTA')
DAEMON_OPTIONS(`Port=587, Name=MSA, M=E')
```

These two lines assign the standard ports to the MTA and the MSA, and a modifier to the MSA. Use the no_default_msa feature to clear the MSA defaults before you set new MSA values with the DAEMON_OPTIONS macro. And then use two DAEMON_OPTIONS commands: the first one for the MTA and the second one for the MSA.

DAEMON_OPTIONS parameters are assigned using *keyword=value* pairs. The possible keywords and values are:

Port The Port keyword assigns a network port number to the daemon. The standard port for an MTA is 25, and the standard port for an MSA is 587. Changing these standard ports means that clients will have difficulty locating the service. The port numbers are therefore rarely changed.

Name The Name keyword identifies the aspect of the sendmail daemon for which the parameters are being set. There are four documented values:

MTA This identifies the traditional Mail Transport Agent interface of sendmail that is used to deliver mail.

MSA This identifies the Mail Submission Agent interface of sendmail that can be used by external MUAs to submit mail. In practice, this function is identical to the MTA function, except for port number, because both aspects of sendmail ensure that all mail, no matter how it arrives, is processed through all necessary rulesets, filters, and

databases.

MTA−v4 This is the same as the MTA interface, and is designed to handle e−mail delivery to hosts with standard 32−bit IPv4 addresses.

MTA−v6 MTA−v6 is an interface designed to handle delivery to hosts that use the 128−bit IPv6 addresses.

Family The Family keyword defines the address family. By default, this is inet, which means that standard IPv4 addresses should be used. An alternate value is inet6, which requests IPv6 addressing.

M The M keyword is a modifier that requests optional processing. M=E turns off the ESMTP ETRN command. This setting is the default for the MSA because it is required by the MSA standard. The M=a setting requires authentication by a trusted authentication method before the MSA will accept the mail message.

LDAP Mail Routing

In addition to the various databases built into sendmail, a Lightweight Directory Access Protocol (LDAP) server can be used with sendmail. If your site uses LDAP for other purposes, you may find some benefit in using it with sendmail. LDAP support is added to sendmail using the following defines, features, and macros:

define(`confLDAP_DEFAULT_SPEC', `*ldap−arguments*') Sets arguments that are required for the LDAP map definition. At a minimum, the name of the LDAP server (−h *server*) and the base distinctive name (−b o=*org*,c=*country*) must be provided. For example:

```
define(`confLDAP_DEFAULT_SPEC', `-h egret.foobirds.org -b o=foobirds.org,c=us')
```

FEATURE(`ldap_routing') Adds the necessary support for LDAP routing to the configuration.

LDAPROUTE_DOMAIN(*domainname*) Adds a domain to the class {LDAPRoute}. Mail routing information for domains in that class is looked up via the LDAP server.

LDAPROUTE_DOMAIN_FILE(*filename*) Identifies the file from which the {LDAPRoute} class is loaded. The file contains a list of the domains for which mail routing information should be obtained from the LDAP server.

This concludes the discussion of m4 macros. The output of all of the files and commands that go into the m4 processor is a sendmail.cf file. The bulk of information about sendmail configuration is found in Chapter 5.

List of Figures

Chapter 1: The Boot Process

Figure 1.1: The boot process flow
Figure 1.2: The SYSV Runlevel Manager Window

Chapter 2: The Network Interface

Figure 2.1: Red Hat's Network Configuration tool
Figure 2.2: The RS–232 hardware handshake
Figure 2.3: kudzu installing a modem driver
Figure 2.4: The Internet Connections window

Chapter 3: Login Services

Figure 3.1: The anonymous FTP RPM

Chapter 4: Linux Name Services

Figure 4.1: A caching–only DNS server RPM

Chapter 5: Configuring a Mail Server

Figure 5.1: sendmail rulesets
Figure 5.2: Contents of the *sendmail–cf* RPM

Chapter 6: The Apache Web Server

Figure 6.1: Linux binaries at the Apache website
Figure 6.2: Enabling Apache with tksysv
Figure 6.3: Apache installation web page
Figure 6.4: A fancy index for /usr/share/doc
Figure 6.5: An invalid certificate warning
Figure 6.6: The CAs built–in Netscape 6.1
Figure 6.7: The Apache server–status display

Chapter 7: Network Gateway Services

Figure 7.1: Circuit switching versus packet switching
Figure 7.2: Routing through networks
Figure 7.3: Contents of the Zebra RPM
Figure 7.4: Installing gated with gnorpm

Chapter 9: File Sharing

Figure 9.1: The Red Hat NFS RPM
Figure 9.2: The Red Hat Samba RPM

Chapter 10: Printer Services

Figure 10.1: Selecting a print queue type

Figure 10.2: The active local printer port
Figure 10.3: Selecting a printer driver
Figure 10.4: Editing a printer configuration
Figure 10.5: Configuring a remote SMB printer
Figure 10.6: Configuring a remote Unix printer

Chapter 11: More Mail Services

Figure 11.1: RPM query of the IMAP package
Figure 11.2: Configuring the mail client
Figure 11.3: Defining Netscape filter rules

Chapter 12: Security

Figure 12.1: Searching the Bugtraq Archives
Figure 12.2: Linux exploits found at http://www.hackers.com/
Figure 12.3: Locating software updates from a vulnerability report
Figure 12.4: Red Hat provides security reports online.
Figure 12.5: The OpenSSH RPM

Chapter 13: Troubleshooting

Figure 13.1: The Kernel Configuration window
Figure 13.2: Network device support configuration options
Figure 13.3: Selecting processor types and features

Appendix A: Installing Linux

Figure A.1: Disk Druid's main screen
Figure A.2: Adding a partition in Disk Druid
Figure A.3: Red Hat firewall configuration
Figure A.4: The Authentication Configuration screen
Figure A.5: Final X configuration window

List of Tables

Chapter 1: The Boot Process

Table 1.1: Valid Action Values

Chapter 2: The Network Interface

Table 2.1: Escape Sequences and Their Meanings

Chapter 4: Linux Name Services

Table 4.1: *named.conf* Configuration Statements
Table 4.2: DNS Database Record Types
Table 4.3: *rndc* Commands
Table 4.4: Databases Controlled by *nsswitch.conf*

Chapter 5: Configuring a Mail Server

Table 5.1: Pattern Matching Symbols
Table 5.2: Rewrite Template Symbols

Chapter 6: The Apache Web Server

Table 6.1: DSO Modules Loaded in the Red Hat Configuration
Table 6.2: Server Side Includes Commands

Chapter 7: Network Gateway Services

Table 7.1: Default *gated* Preference Values

Chapter 8: Desktop Configuration Servers

Table 8.1: pump Command–Line Options

Chapter 9: File Sharing

Table 9.1: Linux mount Command Options
Table 9.2: More mount Options
Table 9.3: smb.conf Variables

Chapter 10: Printer Services

Table 10.1: lpc Commands

Chapter 11: More Mail Services

Table 11.1: POP3 Commands
Table 11.2: IMAP4 Commands
Table 11.3: Access Database Actions
Table 11.4: procmail Recipe Flags

Chapter 12: Security

Table 12.1: Wrapper Variables
Table 12.2: ssh Client Configuration Options

Chapter 13: Troubleshooting

Table 13.1: TCP Protocol States
Table 13.2: tcpdump Packet Filters

Appendix A: Installing Linux

Table A.1: Common Partitions
Table A.2: Single−Character *fdisk* Commands

Appendix B: BIND Reference

Table B.1: BIND 8 Configuration Options
Table B.2: New BIND 9 Options
Table B.3: BIND 8 Logging Categories

Appendix C: The *m4* Macros for *sendmail*

Table C.1: The *sendmail m4* Macros
Table C.2: Optional sendmail Features
Table C.3: OSTYPE defines
Table C.4: Mail Relay defines
Table C.5: MAILER Values

List of Listings

Chapter 1: The Boot Process

Listing 1.1: The Default GRUB Configuration
Listing 1.2: A Sample *lilo.conf* File
Listing 1.3: Adding Password Protection to LILO
Listing 1.4: The *inittab* File
Listing 1.5: Runlevel Initialization Scripts
Listing 1.6: The init.d Script Files
Listing 1.7: Listing Loaded Modules

Chapter 2: The Network Interface

Listing 2.1: Loadable Network Device Drivers
Listing 2.2: An Ethernet Card Configuration Created by *kudzu*
Listing 2.3: A Sample pap−secrets File
Listing 2.4: A Sample chap−secrets File
Listing 2.5: A Sample chat Script

Chapter 3: Login Services

Listing 3.1: An Excerpt of the */etc/protocols* File
Listing 3.2: An Excerpt from */etc/services*
Listing 3.3: Excerpts from an *inetd.conf* File
Listing 3.4: Services Disabled by *inetd*
Listing 3.5: The *xinetd.conf* File
Listing 3.6: The */etc/xinetd.d/wu−ftpd* File
Listing 3.7: Using *chkconfig* to Control *xinetd*
Listing 3.8: A Sample */etc/passwd* File
Listing 3.9: Available Login Shells
Listing 3.10: Examples from the */etc/group* File
Listing 3.11: The Effect of the *useradd* Command
Listing 3.12: Using the *usermod* Command
Listing 3.13: Contents of the */etc/default/useradd* File
Listing 3.14: Contents of the */etc/login.defs* File
Listing 3.15: The *userdel* Command
Listing 3.16: Excerpts of the Red Hat *ftpaccess* File

Chapter 4: Linux Name Services

Listing 4.1: A Sample Host Table
Listing 4.2: A Sample */etc/resolv.conf* File
Listing 4.3: A Sample *zone* Statement
Listing 4.4: A Common Caching−Only Configuration
Listing 4.5: The Red Hat *named.conf* File
Listing 4.6: The Red Hat *localhost.zone* File
Listing 4.7: The *named* Hints File
Listing 4.8: The *named.local* File
Listing 4.9: A DNS Slave Server Configuration
Listing 4.10: A DNS Master Server Configuration
Listing 4.11: A Sample DNS Zone File

Listing 4.12: A DNS Reverse Zone File
Listing 4.13: The Red Hat *rndc.conf* File
Listing 4.14: A Complete *host.conf* File
Listing 4.15: A Sample *nsswitch.conf* File

Chapter 5: Configuring a Mail Server

Listing 5.1: A Sample *aliases* File
Listing 5.2: Sample of the *sendmail.cf* Local Info Section
Listing 5.3: Sample *sendmail.cf* Options
Listing 5.4: *sendmail.cf* Header Commands
Listing 5.5: Sample mailer Definitions
Listing 5.6: Testing the Default *sendmail* Configuration
Listing 5.7: Testing *sendmail* Masquerading
Listing 5.8: The *tcpproto.mc* File
Listing 5.9: The *linux.m4* OSTYPE File
Listing 5.10: The *generic.m4 DOMAIN* File
Listing 5.11: A Customized DOMAIN File
Listing 5.12: A Customized Macro Control File
Listing 5.13: A Sample *genericstable*
Listing 5.14: Testing Address Rewriting

Chapter 6: The Apache Web Server

Listing 6.1: Starting and Checking *httpd*
Listing 6.2: Listing Statically Linked *httpd* Modules
Listing 6.3: Active Directory Containers in Red Hat's *httpd.conf* File
Listing 6.4: Apache Access Controls
Listing 6.5: User Authentication for Web Access
Listing 6.6: Using *mod_auth_db* for User Authentication
Listing 6.7: Adding Users with *dbmmanage*
Listing 6.8: Red Hat's SSL Apache Server Configuration
Listing 6.9: Examining a Certificate with the *openssl* Command
Listing 6.10: Creating an Apache Certificate Signature Request
Listing 6.11: Examining a Certificate Signature Request with *openssl*
Listing 6.12: The Server–Status Location Container

Chapter 7: Network Gateway Services

Listing 7.1: Viewing the *arp* Cache
Listing 7.2: Viewing a Single *arp* Table Entry
Listing 7.3: A Simple Routing Table
Listing 7.4: A sample */etc/gateways* file
Listing 7.5: Sample *zebra.conf* File
Listing 7.6: Examining *zebra.conf* through the *vtysh* Interface
Listing 7.7: The Port Numbers Used by the Zebra Suite
Listing 7.8: Reconfiguring *zebra.conf* through the *vtysh* Interface
Listing 7.9: A Sample *ripd.conf* File
Listing 7.10: A *zebra.conf* File for a Linux Host
Listing 7.11: A *zebra.conf* File for a RIP/OSPF Router
Listing 7.12: A *ripd.conf* File for a RIP/OSPF Router
Listing 7.13: A Sample *ospfd.conf* File

Listing 7.14: A Sample *bgpd.conf* File
Listing 7.15: A *gated* RIPv2 Configuration
Listing 7.16: A *gated* OSPF/RIPv2 Interior Router Configuration
Listing 7.17: A *gated* OSPF/BGP Exterior Router Configuration

Chapter 8: Desktop Configuration Servers

Listing 8.1: A Sample *dhcpd.conf* File
Listing 8.2: A Sample *dhcpcd–eth0.info* File
Listing 8.3: A Sample *ifcfg–eth0* File
Listing 8.4: A Sample *pump.conf* File
Listing 8.5: A Sample *dhclient.conf* File

Chapter 9: File Sharing

Listing 9.1: Examining File Permissions with *ls*
Listing 9.2: Displaying RPC Ports
Listing 9.3: A Sample */etc/exports* File
Listing 9.4: The *showmount* Command
Listing 9.5: Sample Mount Commands
Listing 9.6: A Sample *fstab* File
Listing 9.7: A Sample */etc/mtab* File
Listing 9.8: A Sample *lmhosts* File
Listing 9.9: Active Lines in the Red Hat *smb.conf* File
Listing 9.10: Samba File Shares
Listing 9.11: Using *smbclient*
Listing 9.12: Checking */proc/filesystems*
Listing 9.13: An *smbmount* Example

Chapter 10: Printer Services

Listing 10.1: Listing the Printer Ports
Listing 10.2: A Sample *printcap* File
Listing 10.3: Using *lpc* Interactively
Listing 10.4: Viewing and Reordering a Print Queue
Listing 10.5: Removing Jobs from the Print Queue
Listing 10.6: *smb.conf* with Printer Sharing
Listing 10.7: The *script.cfg* File for a Samba Printer

Chapter 11: More Mail Services

Listing 11.1: Using the POP Protocol with *telnet*
Listing 11.2: Testing IMAP with *telnet*
Listing 11.3: Permitting Mail Relaying
Listing 11.4: Testing the *dnsbl* Feature
Listing 11.5: A Sample Access Database for *sendmail*
Listing 11.6: Adding the Access Database to the Configuration
Listing 11.7: A *Local_check_mail* Example
Listing 11.8: An Example of Creating a Local Ruleset
Listing 11.9: A sample *.procmailrc* file

Chapter 12: Security

Listing 12.1: The *tcpd* Security Log
Listing 12.2: An *xinetd* Configuration File
Listing 12.3: *xinetd.conf* Access Controls
Listing 12.4: Sample *iptables* Commands
Listing 12.5: Linux Rejects Weak Passwords
Listing 12.6: Excerpts from the Shadow Password File
Listing 12.7: Modifying */etc/shadow* with *usermod*
Listing 12.8: Generating OPIE Password Phrases
Listing 12.9: A Sample *ssh* Login
Listing 12.10: An Example of the *ssh−keygen* Command
Listing 12.11: The Red Hat *sshd_config* file
Listing 12.12: The Red Hat *ssh_config* file

Chapter 13: Troubleshooting

Listing 13.1: Adding the New Kernel to *lilo.conf*
Listing 13.2: Adding a New Kernel to *grub.conf*
Listing 13.3: Red Hat Network Interface Configuration Files
Listing 13.4: Displaying the Configuration with *ifconfig*
Listing 13.5: Viewing the ARP Table
Listing 13.6: The *arpwatch arp.dat* File
Listing 13.7: Sample *arpwatch* E−mail Reports
Listing 13.8: Testing a PPP Link with *minicom*
Listing 13.9: A Successful *ping* Test
Listing 13.10: A Failed *ping* Test
Listing 13.11: Displaying the Routing Table
Listing 13.12: Testing a Route with *traceroute*
Listing 13.13: Displaying Network Socket Connections
Listing 13.14: Display All Sockets
Listing 13.15: A *telnet* Handshake as Seen by *tcpdump*
Listing 13.16: Monitoring Traffic with *tcpdump*
Listing 13.17: Testing DNS with *nslookup*
Listing 13.18: Testing Continues
Listing 13.19: Testing DNS with the *host* Command
Listing 13.20: Testing DNS with *dig*

Appendix A: Installing Linux

Listing A.1: Using *rawrite*
Listing A.2: Creating Floppy Disks with *dd*
Listing A.3: Partitioning with *fdisk*
Listing A.4: Adding Logical Partitions
Listing A.5: Assigning Filesystem Types

Appendix B: BIND Reference

Listing B.1: The BIND 8 *options* Statement Syntax
Listing B.2: The BIND 9 *options* Statement Syntax
Listing B.3: BIND 8 *logging* Command Syntax
Listing B.4: BIND 9 *logging* Command Syntax
Listing B.5: BIND 8 *zone* Statement Syntax
Listing B.6: BIND 9 *zone* Statement Syntax

Listing B.7: The BIND 8 *server* Statement Syntax
Listing B.8: The BIND 9 *server* Statement Syntax
Listing B.9: The *key* Statement Syntax
Listing B.10: The *acl* Statement Syntax
Listing B.11: The *trusted–keys* Statement Syntax
Listing B.12: BIND 8 *controls* Statement Syntax
Listing B.13: BIND 9 *controls* Statement Syntax
Listing B.14: The *view* Statement Syntax

List of Sidebars

Introduction

Sidebars

Chapter 2: The Network Interface

Address Mask, Subnet Mask, or Network Mask?

Chapter 4: Linux Name Services

Resolver Timeouts

Chapter 7: Network Gateway Services

Proxy ARP
Counting to Infinity

Chapter 8: Desktop Configuration Servers

Using dhcpd with Old Linux Kernels
Placing DHCP Servers

Chapter 9: File Sharing

Hidden Bits
Coordinating UIDs and GIDs
Clear–Text Password

Chapter 11: More Mail Services

Spam, Spam, Spam, Spam, and Spam

Chapter 12: Security

Realistic Wrapper Rules
Password Dos and Don'ts
The OPIE Transition Mechanism

Chapter 13: Troubleshooting

Adapter Card Configuration

Appendix A: Installing Linux

Working with a Windows Partition
Symbolic Links