Third Edition

The Official ubuntu Server Book

Kyle Rankin
Benjamin Mako Hill

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Praise for *The Official Ubuntu Server Book*

"Murphy's Law is never truer than when it comes to administering a Linux server. You can pretty much count on something happening to your machine at a time when you need it the most. That's when a book with some basic troubleshooting instructions is worth every penny you paid for it. Chapter 11 covers the steps you should take when something goes wrong."

—Paul Ferrill, LinuxPlanet.com reviewer

"College-level collections catering to Linux programmers and developers will find *The Official Ubuntu Server Book,* a top addition to the collection, covering a complete, free server operating system in a guide to getting going quickly. From making the most of Ubuntu Server's latest technologies to automating installs and protecting the server using Ubuntu's built-in security tools, *The Official Ubuntu Server Book,* is packed with keys to success for any Ubuntu user."

—Jim Cox, *Midwest Book Review*

"This book will get you started on the path of the server admin, both within the context of Ubuntu server and in the larger realm of managing a server infrastructure. The desktop and server versions of Ubuntu are continuing to mature. Read this book if you want to keep up."

—James Pyles, author

The Official Ubuntu Server Book

Third Edition

Kyle Rankin
Benjamin Mako Hill

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/ph

Cataloging-in-Publication Data is on file with the Library of Congress.

Copyright © 2014 Canonical, Ltd.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax: (617) 671-3447

The Introduction and Chapter 3 of this book are published under the Creative Commons Attribution-ShareAlike 3.0 license, http://creativecommons.org/licenses/by-sa/3.0/.

ISBN-13: 978-0-13-301753-3
ISBN-10: 0-13-301753-2
Text printed in the United States on recycled paper at Edwards Brothers Malloy in Ann Arbor, Michigan.
First printing, July 2013

I dedicate this book to my wife, Joy. It is not easy to balance a full-time job and writing a book while still having time for a family. She has endured many a book-writing process at this point and has always been my main source of support and motivation.

—Kyle Rankin

Contents at a Glance

Contents	*ix*
Preface	*xix*
Acknowledgments	*xxv*
About the Authors	*xxvii*
Introduction	*xxix*
Chapter 1: Installation	1
Chapter 2: Essential System Administration	17
Chapter 3: Package Management	51
Chapter 4: Automated Ubuntu Installs	83
Chapter 5: Guide to Common Ubuntu Servers	125
Chapter 6: Security	199
Chapter 7: Backups	239
Chapter 8: Monitoring	267
Chapter 9: Virtualization and Cloud Computing	297
Chapter 10: Fault Tolerance	341
Chapter 11: Troubleshooting	399
Chapter 12: Rescue and Recovery	429
Chapter 13: Help and Resources	449
Chapter 14: Basic Linux Administration	463
Appendix: Cool Tips and Tricks	485
Index	495

Contents

Preface xix

Acknowledgments xxv

About the Authors xxvii

Introduction xxix

Welcome to Ubuntu Server xxix
Free Software, Open Source, and Linux xxx
Free Software and GNU xxxi
Linux xxxii
Open Source xxxiii
A Brief History of the Ubuntu Project xxxiv
Mark Shuttleworth xxxiv
The Warthogs xxxvi
What Does *Ubuntu* Mean? xxxvii
Creating Canonical xxxviii
The Ubuntu Community xxxix
Ubuntu Promises and Goals xli
Philosophical Goals xli
Conduct Goals and Code of Conduct xliii
Technical Goals xliv
Canonical and the Ubuntu Foundation xlvi
Canonical, Ltd. xlvi
Canonical's Service and Support xlvii
The Ubuntu Foundation xlviii
History of Ubuntu Server xlix
Simple, Secure, Supported li

CHAPTER 1 Installation 1

Get Ubuntu 2
Boot Screen 3

Disk Partitioning 5
What Is a Partition? 5
Guided—Use Entire Disk 8
Guided with LVM 8
Manual 9
Server Roles 13
Installer Console 15
Reboot the System 16

CHAPTER 2 Essential System Administration 17

Basic Command-Line Administration 18
Move Around the System 18
File Ownership 21
Check Running Processes 21
Edit Files 23
Become Root 24
Ubuntu Boot Process 24
GRUB 25
The Kernel Boot Process 26
/sbin/init 27
Services 34
File System Hierarchy 39
Networking 45
Network Configuration Files 46
Core Networking Programs 48

CHAPTER 3 Package Management 51

Introduction to Package Management 52
Background on Packages 53
What Are Packages? 53
Basic Functions of Package Management 55
Advanced Functions of Package Management Systems 58
Debian Packages 60
Source Packages 60
Binary Packages 63
Package Management in Ubuntu 63
Staying Up-to-Date 64
Searching and Browsing 65
Installation and Removal 67
Manipulating Installed Packages 69
Manipulating Repositories 71

Ubuntu Default Repositories 73
Using Other Repositories 74
Upgrading a Whole System 75
Mirroring a System 76
Making Your Own Packages 77
Rebuilding Packages 77
New Upstream Versions 79
Building Packages from Scratch 80
Hosting Your Own Packages 81

CHAPTER 4 Automated Ubuntu Installs 83

Preseeding 84
Basic Preseed Configuration for CD-ROM 85
Networking Options 89
Partitioning 91
Packages and Mirrors 96
User Settings 98
GRUB 99
Miscellaneous 100
Dynamic Preseeding 100
Kickstart 104
Basic Kickstart Configuration for CD-ROM 104
Changes and Limitations in Ubuntu Kickstart 108
Run Custom Commands during the Install 110
PXE Boot Server Deployment 111
DHCP 112
TFTPD 113
Configure Pxelinux 113
Web 116
Test Your PXE Server 116
Customize Automated Installs 118
Multiple Kickstart Files 118
Boot Cheat Codes 119
DHCP Selection 121
DHCP Selection by Subnet 123

CHAPTER 5 Guide to Common Ubuntu Servers 125

DNS Server 126
Install BIND 127
Ubuntu Conventions 127
Caching Name Server 129

DNS Master 129
DNS Slave 132
Manage BIND with rndc 134
Web Server 135
Install a Web Server 135
Ubuntu Apache Conventions 136
apache2ctl 139
Apache Documentation 141
WordPress, a Sample LAMP Environment 141
Mail Server 144
Install Postfix 144
Postfix Configuration Types 145
Ubuntu Postfix Conventions 146
Administering Postfix 148
Default Postfix Example 150
Secondary Mail Server 153
Greylisting Mail Server 154
POP/IMAP Server 156
Enable Maildirs on Postfix 156
Install Dovecot 157
Ubuntu Dovecot Conventions 158
OpenSSH Server 158
Ubuntu OpenSSH Conventions 159
DHCP Server 160
Install DHCP 160
Ubuntu DHCP Conventions 161
Configure DHCP 161
Database Server 163
MySQL 163
PostgreSQL 168
File Server 174
Samba 174
NFS 177
Edubuntu and LTSP 180
What Is LTSP? 180
Technical Details of the LTSP Boot Process 181
The Benefits of LTSP 182
Other Uses 183
LTSP Availability in Ubuntu 183
Installing an LTSP Server 183

LTSP Server Configurations 184
The Installation Procedure 186
Initial LTSP Server Setup 188
Initial LTSP Client Setup 189
Installing the LTSP Environment in Ubuntu or on a Desktop Installation 190
Special LTSP Cases 191
Changing Your IP Address 194
Local Devices over LTSP 195
Sound over LTSP 197

CHAPTER 6 Security 199

General Security Principles 200
Sudo 201
Configure sudo 203
sudo Aliases 205
AppArmor 206
AppArmor Profiles 207
Enforce and Complain Modes 209
Ubuntu AppArmor Conventions 210
SSH Security 210
sshd_config 211
Key-Based Authentication 211
SSH Brute-Force Attacks 213
Firewalls 214
ufw Commands 216
ufw Rule Syntax 217
Extended ufw Rules 218
ufw Examples 220
Ubuntu ufw Conventions 224
Intrusion Detection 226
Update Tripwire Policy 227
Initialize the Tripwire Database 229
Update the Tripwire Database 230
Ubuntu Tripwire Conventions 232
Incident Response 233
Do You Prosecute? 233
Pull the Plug 233
Image the Server 234
Server Redeployment 234
Forensics 235

CHAPTER 7 Backups 239

Backup Principles 240
Drive Imaging 242
Database Backups 244
MySQL 244
PostgreSQL 248
BackupPC 249
BackupPC Storage 250
Default BackupPC Configuration 251
Configure the Client Machine 254
Add the Client to BackupPC 255
Start the First Backup Job 256
rsync Tweaks 258
Restore Files 263
Ubuntu BackupPC Conventions 265

CHAPTER 8 Monitoring 267

Local Monitoring Tools 268
Smartmontools 268
sysstat 269
Ganglia 273
Install ganglia-monitor on All Hosts 274
Configure Ganglia Server 276
Install the Ganglia Web Front End 278
Nagios 280
Install GroundWork 281
GroundWork File Conventions 282
Initial Configuration 283
Configure Nagios 286
Commit Changes to Nagios 289
Configure Contact List 289
Enable Notifications for Nagios 290
Add a Service Check to a Host 291
Add a New Host 291
Advanced Configuration 292
More GroundWork Information 296

CHAPTER 9 Virtualization and Cloud Computing 297

KVM 298
Install KVM 298
Enable Support in BIOS 299

Install KVM Packages 299
Configure KVM Networking 300
Create a New VM 302
Extra vmbuilder Options 306
Manage VMs with virsh 309
KVM Graphical Console and Management Tools 312
Amazon EC2 315
Register an Account 315
Setting Up EC2 API Tools 316
Create an ssh Key Pair 319
Pick an Amazon AMI 320
Security Groups 324
SSH into the Instance 326
Start, Stop, and Terminate an Instance 327
Userdata Scripts 328
Juju 330
Install and Configure Juju 330
Juju Bootstrap 333
Deploy Juju Services 333
Fault Tolerance 337
Destroying Instances 338

CHAPTER 10 Fault Tolerance 341

Fault Tolerance Principles 342
RAID 344
RAID Levels 345
Configure RAID during Installation 346
Configure RAID after Installation 348
Software RAID Management 351
Migrate Non-RAID to Software RAID 354
Migrate from RAID 1 to RAID 5 359
Add a Drive to a RAID 5 Array 366
LVM 369
LVM Theory and Jargon 370
Setting Up LVM 371
Ethernet Bonding 372
Ubuntu 10.04 Network Configuration 375
Ubuntu 12.04 and Newer Network Configuration 376
Enable the Bonded Interface 377

Clusters 378
Heartbeat 380
DRBD 388

CHAPTER 11 Troubleshooting 399

General Troubleshooting Philosophy 400
Divide the Problem Space 400
Favor Quick, Simple Tests over Slow, Complex Tests 401
Favor Past Solutions 401
Good Communication Is Critical When Collaborating 402
Understand How Systems Work 402
Document Your Problems and Solutions 402
Use the Internet, but Carefully 403
Resist Rebooting 403
Localhost Troubleshooting 403
Host Is Sluggish or Unresponsive 404
Out of Disk Space 413
Network Troubleshooting 416
Server A Can't Talk to Server B 416
Can I Route to the Remote Host? 421
Test the Remote Host Locally 424
Hardware Troubleshooting 425
Network Card Errors 425
Test Hard Drives 426
Test RAM 427

CHAPTER 12 Rescue and Recovery 429

Ubuntu Recovery Mode 430
File Systems Won't Mount 432
Problem Init Scripts 434
Reset Passwords 435
Ubuntu Server Recovery CD 435
Boot into the Recovery CD 436
Recover GRUB 438
Repair the Root File System 438
Ubuntu Desktop Live CD 439
Boot the Live CD 439
Add the Universe Repository 439
Recover Deleted Files 440

Restore the Partition Table 443
Rescue Dying Drives 444

CHAPTER 13 Help and Resources 449

Paid Support from Canonical 450
Forums 451
Internet Relay Chat 452
Mailing Lists 455
Online Documentation 456
Localhost Documentation 457
Local Community Teams 458
Other Languages 459
Tech Answers System (Launchpad) 459
Bug Reporting 459
For More Information 461

CHAPTER 14 Basic Linux Administration 463

Shell Globs 464
Regular Expressions 465
Pipes and Redirection 466
Redirection 470
File Permissions and Ownership 472
chmod 474
Linux File Types 474
Symbolic Links 475
Hard Links 476
Device Files 477
At and Cron 478
At 478
Cron 480

APPENDIX Cool Tips and Tricks 485

Avoid That grep Command in grep Output 485
Shortcut to a Command Path 486
Wipe a Drive in One Line 486
Run a Command Over and Over 487
Make a Noise When the Server Comes Back Up 487
Search and Replace Text in a File 487
find and exec Commands 488

Bash Commands with Too Many Arguments 488
Use Your Bash History 489
Are These Files Identical? 489
Go Back to Your Previous Directory 489
Find Out Who Is Tying Up a File System You Want to Unmount 490
Send a Test E-mail Using telnet 490
Easy SSH Key Sharing 491
Get the Most Out of Dig 492

Index **495**

Preface

WELCOME to the third edition of *The Official Ubuntu Server Book*!

When most people talk about Ubuntu these days, they tend to talk about the Ubuntu Desktop. After all, it's the easy-to-use, "just works" approach to the desktop that has made Ubuntu one of the most popular desktop Linux distributions. What has gotten less attention, although even that is starting to change, is Ubuntu Server. It turns out that desktop Linux users aren't the only ones who want their distribution to "just work"—system administrators appreciate that on their servers as well. In Ubuntu Server you will find all of the powerful server infrastructure from the Debian project plus that extra bit of Ubuntu polish, innovation, and focus on ease of use.

About This Book

This book is the result of the collaborative effort of not just the principal authors, but of the Ubuntu Server team itself. As it is the official, authorized book on Ubuntu Server, the focus has been on a server guide based on our collective experience. Beyond that, the goal is to have something to offer to both the beginner system administrator and the battle-hardened senior sysadmin. On the surface it might seem a tough balance to achieve, but in reality both groups ultimately want the same thing: for their servers to work. Now it's true that some administrators revel in doing things the hard way. Some even treat it as a point of pride. The thing is, all of us who have administered servers for years can do and have done things the hard way as well, but ultimately you realize that there's nothing particularly impressive in doing everything by hand—in the end you just have too much to do and any time-saving steps are welcome.

As you will see, most of this book takes a pragmatic approach to server management. Where Ubuntu offers new programs or features to ease administration and save time, you will find them mentioned here. If you are a beginner administrator, you will find that administering an Ubuntu server isn't nearly as difficult as you might think. Experienced administrators, especially those coming from other platforms, will find numerous time-saving tips and programs, as well as where Ubuntu has updated how a service is organized (Apache being a good example); you can treat this book as a map to point you to all of the right directories.

One great thing about Ubuntu as a server is that there are so many great server packages available for it. Of course, this creates a dilemma for us as writers: It's just not possible to feature every available e-mail and IMAP/POP3 server, for instance. In these cases we've tried to pick out programs that are easy to install, configure, and use under Ubuntu, as well as highlight programs that are preferred by the authors and server team. While doing that, there's a good chance that your favorite program for X, Y, and Z was left out. It's certainly no slight against any of those programs—we just had to draw the line somewhere.

How the Book Is Organized

Different people read tech books differently. Some people read them cover to cover, and others skip right ahead to the topic they need immediate help with. You will find that the way this book is organized lends itself well to both approaches. The first few chapters lay the foundation so you can install Ubuntu and navigate the system even if it's your first time. After that the chapters focus on particular server topics, from security to monitoring to system rescue.

- Chapter 1—Installation. In the first chapter you will learn how to use the default Ubuntu Server CD to install Ubuntu on a server. This guide includes a complete walk-through of the installation process from the initial boot screen to partitioning to your first login prompt.
- Chapter 2—Essential System Administration. If you are new to Ubuntu system administration, the amount of learning ahead of you might seem daunting. In this chapter you will find not only a solid foundation of instructions on how to navigate the Linux command

line, but also an introduction to the Ubuntu boot process and the standards behind all of the directories on an Ubuntu system. By the end of the chapter you should have a good basis to continue with the rest of the book.

- Chapter 3—Package Management. This chapter introduces you to packages and the packaging system—the way that Ubuntu handles the installation, removal, and management of software. We provide a solid foundation in what packages do and how they do it before drilling down into the details of how an administrator can manage software the Ubuntu way. In the final pages, we cover the way that administrators can switch from being consumers to producers and begin making their own packages.
- Chapter 4—Automated Ubuntu Installs. While you can certainly install Ubuntu step by step from the install CD, that method doesn't work so well when you have tens or hundreds of servers to install. This chapter covers the preseed method for automating Ubuntu installs along with Kickseed—Ubuntu's port of Kickstart. In addition to a description of how to use both of these technologies independently, you will find out it's even better when you use them together.
- Chapter 5—Guide to Common Ubuntu Servers. There is an enormous number of services you can run on an Ubuntu server. In this chapter we highlight some of the more popular services, from Web to e-mail to file services. If you are a new administrator, you will find a simple guide on how to install and configure these services for the first time. If you are an experienced administrator coming from another distribution, you will find this chapter a handy, how-to guide on how Ubuntu organizes all of the configuration files for your favorite services.
- Chapter 6—Security. Security is an important topic for any administrator. Ubuntu Server already is pretty secure by default, and in this chapter we highlight some of these mechanisms, along with steps you can take to increase your security even further. Some of the security topics include `sudo`, firewall configuration, an introduction to forensics, and even Ubuntu's AppArmor software.
- Chapter 7—Backups. There are two kinds of administrators: those who back up their servers and those who haven't lost valuable data yet. Backup software abounds for Linux as a whole and for Ubuntu

specifically, and in this chapter you will see a few easy-to-set-up approaches to keeping your data secure.

- Chapter 8—Monitoring. Monitoring is one of the most valuable systems an administrator can set up while simultaneously being the most annoying (why do servers always seem to page you in the middle of the night?). In this chapter we cover some different approaches to monitoring systems both for trending purposes and to alert you to any problems. By the end of the chapter you will no longer lose sleep wondering if a server is up—you'll lose sleep only when it goes down.

- Chapter 9—Virtualization and Cloud Computing. Virtualization and cloud computing are hot topics in system administration today. With increasingly powerful hardware out there, virtualization provides you with a way to squeeze the most efficiency out of your servers and cloud computing abstracts even further so that servers can be created and destroyed on a whim. In this chapter we cover one of the most popular server-based virtualization tools out there: KVM. We also cover how to use Amazon's EC2 cloud environment with command-line tools and also how to automate EC2 server deployment with a new Canonical tool called Juju.

- Chapter 10—Fault Tolerance. If a lot is riding on your servers and your downtime is measured in dollars and not minutes, you realize very quickly that your servers need fault tolerance. This chapter covers the Ubuntu software RAID, including steps to migrate from one type of RAID to another. Then we will cover how to set up redundant network connections and finish up with a guide to setting up your own Linux cluster. We also discuss how to get up and running with logical volume management (LVM).

- Chapter 11—Troubleshooting. No matter how great an administrator you are, eventually something on your servers will fail. Over the years you develop a series of troubleshooting steps you go through whenever you find a problem on your systems. In this chapter we condense years of troubleshooting experience into a series of step-by-step guides to walk you through common server and network problems and how to use standard Ubuntu tools and techniques to diagnose them.

- Chapter 12—Rescue and Recovery. We've often said that we've learned more about Linux from fixing a broken system than in any other way. In some environments when a system won't boot, an administrator might just install a new operating system. Under Ubuntu, however, you'll find that most common boot problems also have a common, easy solution. In this chapter we discuss how to use different stages of rescue modes both on Ubuntu and the Ubuntu Server install CD itself to repair your system.
- Chapter 13—Help and Resources. One great thing about Ubuntu is just how many support avenues there are when you need help. Whether it's documentation on the machine itself, guides on the official Ubuntu site, forums, or even professional Canonical support, when you are stuck you aren't alone. In this chapter we cover all of the different ways to get support for your Ubuntu server.
- Chapter 14—Basic Linux Administration. This chapter picks up where Chapter 2, Essential System Administration, left off. Here we discuss some of the core foundation concepts behind Linux administration, including file permissions, different file types, pipes, and other core Linux information. Beginner administrators will find this a very useful guide to flesh out any gaps in their command-line knowledge, and the experienced administrators will find it a good refresher on core concepts.
- Appendix—Cool Tips and Tricks. Over the years you develop all sorts of useful tips, one-liners, and other shell commands that make your life as an administrator easier. Here you will find some of our favorite time-saving tips and hacks in rapid-fire form.

Media with This Book

This book includes two versions of Ubuntu Server: Ubuntu 12.04.2 for 64-bit machines and the latest Ubuntu 13.04 release, so you can pick the version that best matches your needs.

Although we have included both Ubuntu 12.04.2 and 13.04 releases and have written the book for both versions, you might decide to try out a

newer Ubuntu release. In that case, just go to http://ubuntu.com and either download the CD image or request a copy to be sent to you. No matter which Ubuntu Server CD you pick, it's relatively easy to use the CDs. Just insert the version you want to install into your computer and boot from the CD-ROM. When the CD boots, you will see a number of options on the screen, but to install Ubuntu Server, just select Install Ubuntu Server. The installer that launches will ask some fairly straightforward questions common to most install discs, and if you get stuck, just turn to Chapter 1 for a more in-depth walk-through of the install process.

Acknowledgments

JORGE, I WOULDN'T HAVE been involved in this book if it weren't for you. I'm one in a long list of people using Ubuntu because of Jorge. His enthusiasm is infectious, and I can't count how many times he's introduced me to some cool new program or tool that I write off at first and then somehow find myself using eventually.

Debra and Mako, it has been great working with both of you on this project, and thank you for the opportunity and guidance. Also thanks to Matthew for his help on the support chapter. Robert, thanks so much for your great attention to detail and tracking down all the areas where I had made typos and mistakes. Thanks to Bill "The Cloud" Childers for providing me with equipment for the UEC section.

Extra thanks to Dustin, Nick, Jamie, Kees, Alan, Mathias, Thierry, and the rest of the Ubuntu Server team for all of your excellent feedback and help through this process.

—Kyle Rankin

About the Authors

Kyle Rankin is a senior systems administrator, the author of *DevOps Troubleshooting*, *Knoppix Hacks*, *Knoppix Pocket Reference*, *Linux Multimedia Hacks*, and *Ubuntu Hacks*, and he has contributed to a number of other O'Reilly books. Kyle is also an award-winning columnist for *Linux Journal* and has had articles featured in *PC Magazine*, *TechTarget*, and other publications.

Benjamin Mako Hill is a Ph.D. candidate at the Sloan School of Management and Media Lab at MIT, and, as of Fall 2013, an assistant professor of communications at the University of Washington. As part of the founding Ubuntu team, his charge at Canonical was to help grow the Ubuntu development and user community during the project's first year. Mako has continued his involvement with Ubuntu as a member of the Community Council governance board and through projects such as this book.

Introduction

THIS INTRODUCTION GIVES AN overview of Ubuntu and Ubuntu Server. After a quick welcome, it includes a brief history of free software, open source, and GNU/Linux and of the Ubuntu project itself, with a focus on some of the major players on the Ubuntu scene. This introduction ends where the rest of this book will continue: with a history of the Ubuntu Server project and an overview of that project's goals and accomplishments.

Welcome to Ubuntu Server

In the just over eight years of its life, Ubuntu has become one of the most popular GNU/Linux-based operating systems. In the process, however, public perception has been disproportionately focused on Ubuntu's role as a desktop-based operating system. While all popularity is certainly welcome for those of us involved in the project, this success has, at times, overshadowed the rock-solid server operating system that Ubuntu has been constructed to be. For those of us who have helped build out Ubuntu's server-specific features and who use it daily, this is both unfortunate and undeserved. Designed and used as a server since day one, Ubuntu has supported a server team that was one of the first active teams in the Ubuntu community and has been one of the most successful. Although perceptions have changed in large part, many prospective users—and even some current Ubuntu users—often continue to think of Ubuntu as something for desktops.

Perhaps it is just that people are so surprised at the usability of Ubuntu on the desktop—especially in the early days when expectations for desktop GNU/Linux distributions were low—that the public focus naturally has drifted away from Ubuntu's server offering. Lots of other GNU/Linux distributions run great on servers, but a solid desktop experience continues

to be surprising to many users. As a result, when people talk about Ubuntu, they often tend to talk about desktops. Perhaps, on the other hand, people just figured that such a well-polished desktop must have come at the cost of the server-oriented features and support. Of course, no such sacrifices were made.

To a large extent, times have changed. The Ubuntu Server team has continued its tireless work both to improve the experience for server users of Ubuntu and to help promote Ubuntu as a server solution. Documentation, testimonials, certification of server-based software, support contracts from a variety of sources, training courses, and more have all contributed to remaking Ubuntu into a powerful player on the server. Although its desktop credentials have not been diminished, Ubuntu's server chops are increasingly difficult to overlook. Over the past two years, Ubuntu has begun to become a major player in the GNU/Linux server market.

More than anything else, testimonials have spread and the small group of early Ubuntu Server users has spread the word. More and more people choose Ubuntu for their servers every day. In fact, this book is simply the latest striking example of just how far Ubuntu on servers has come. Not only do people now know that Ubuntu runs on a server, they know it runs well. This book is publishable only because there is a market for it. That market is made up of people who have heard good things about Ubuntu on the server and who are getting ready to take the plunge themselves. Welcome. We hope we can help make the process easier. We've come a long way, and we're still only just beginning.

Free Software, Open Source, and Linux

A history of Ubuntu Server must, in large part, be a history of Ubuntu itself. A history of Ubuntu must, in large part, be a history of the free software movement and of the Linux kernel. While thousands of individuals have contributed in some form to Ubuntu, the project has succeeded only through the contributions of many thousands more who have indirectly laid the technical, social, and economic groundwork for Ubuntu's success. While introductions to free software, open source, and GNU/Linux can be found in many other places, no introduction to Ubuntu is complete with-

out a brief discussion of these concepts and the people and history behind them. It is around these concepts and within these communities that Ubuntu was motivated and born. Ultimately, it is through these ideas that it is sustained.

Free Software and GNU

In a series of events that have almost become legend through constant repetition, Richard M. Stallman created the concept of free software in 1983. Stallman grew up with computers in the 1960s and 1970s, when computer users purchased very large and extremely expensive mainframe computers, which were then shared among large numbers of programmers. Software was, for the most part, seen as an add-on to the hardware, and every user had the ability and the right to modify or rewrite the software on his or her computer and to freely share this software. As computers became cheaper and more numerous in the late 1970s, producers of software began to see value in the software itself. Producers of computers began to argue that their software was copyrightable and a form of intellectual property much like a music recording, a film, or a book's text. They began to distribute their software under licenses and in forms that restricted its users' abilities to use, redistribute, or modify the code. By the early 1980s, restrictive software licenses had become the norm.

Stallman, then a programmer at MIT's Artificial Intelligence Laboratory, became increasingly concerned with what he saw as a dangerous loss of the freedoms that software users and developers had up until that point enjoyed. He was concerned with computer users' ability to be good neighbors and members of what he thought was an ethical and efficient computer-user community. To fight against this negative tide, Stallman articulated a vision for a community that developed liberated code—in his words, "free software." He defined free software as software that had the following four characteristics—labeled as freedoms 0 through 3 instead of 1 through 4 as a computer programmer's joke:

- The freedom to run the program for any purpose (freedom 0)
- The freedom to study how the program works and adapt it to your needs (freedom 1)

- The freedom to redistribute copies so you can help your neighbor (freedom 2)
- The freedom to improve the program and release your improvements to the public so that the whole community benefits (freedom 3)

Access to source code—the human-readable and modifiable blueprints of any piece of software that can be distinguished from the computer-readable version of the code that most software is distributed as—is a prerequisite to freedoms 1 and 3. In addition to releasing this definition of free software, Stallman began a project with the goal of creating a completely free OS to replace the then-popular UNIX. In 1984, Stallman announced this project and called it GNU—another joke in the form of a recursive acronym for "GNU's Not UNIX."

Linux

By the early 1990s, Stallman and a collection of other programmers working on GNU had developed a near-complete OS that could be freely shared. They were, however, missing a final essential piece in the form of a kernel—a complex system command processor that lies at the center of any OS. In 1991, Linus Torvalds wrote an early version of just such a kernel, released it under a free license, and called it Linux. Linus's kernel was paired with the GNU project's development tools and OS and with the graphical windowing system called X. With this pairing, a completely free OS was born—free both in terms of price and in Stallman's terms of freedom.

All systems referred to as Linux today are, in fact, built on the work of this collaboration. Technically, the term Linux refers only to the kernel. Many programmers and contributors to GNU, including Stallman, argue emphatically that the full OS should be referred to as GNU/Linux in order to give credit not only to Linux but also to the GNU project and to highlight GNU's goals of spreading software freedom—goals not necessarily shared by Linus Torvalds. Many others find this name cumbersome and prefer calling the system simply Linux. Yet others, such as those working on the Ubuntu project, attempt to avoid the controversy altogether by referring to GNU/Linux only by using their own project's name.

Open Source

Disagreements over labeling did not end with discussions about the naming of the combination of GNU and Linux. In fact, as the list of contributors to GNU and Linux grew, a vibrant world of new free-software projects sprouted up, facilitated in part by growing access to the Internet. As this community grew and diversified, a number of people began to notice an unintentional side effect of Stallman's free software. Because free software was built in an open way, *anyone* could contribute to software by looking through the code, finding bugs, and fixing them. Because software ended up being examined by larger numbers of programmers, free software was higher in quality, performed better, and offered more features than similar software developed through proprietary development mechanisms. In many situations, the development model behind free software led to software that was *inherently better* than proprietary alternatives.

As the computer and information technology industry began to move into the dot-com boom, one group of free software developers and leaders, spearheaded by two free software developers and advocates—Eric S. Raymond and Bruce Perens—saw the important business proposition offered by a model that could harness volunteer labor or interbusiness collaboration and create intrinsically better software. However, they worried that the term *free software* was problematic for at least two reasons. First, it was highly ambiguous—the English word *free* means both gratis, or at no cost (e.g., as in "free beer") and liberated in the sense of freedom (e.g., as in "free speech"). Second, there was a feeling, articulated most famously by Raymond, that all this talk of freedom was scaring off the very business executives and decision makers whom the free software movement needed to impress in order to succeed.

To tackle both of these problems, this group coined a new phrase—*open source*—and created a new organization called the Open Source Initiative. The group set at its core a definition of open source software that overlapped completely and exclusively both with Stallman's four-part definition of free software and with other community definitions that were also based on Stallman's.

One useful way to understand the split between the free software and open source movements is to think of it as the opposite of a schism. In religious

schisms, churches separate and do not work or worship together because of relatively small differences in belief, interpretation, or motivation. For example, most contemporary forms of Protestant Christianity agree on *almost* everything but have separated over some small but irreconcilable difference. However, in the case of the free software and open source movements, the two groups have fundamental disagreements about their motivation and beliefs. One group is focused on freedom, while the other is focused on pragmatics. Free software is most accurately described as a social movement, whereas open source is a development methodology. However, the two groups have no trouble working on projects hand in hand.

In terms of the motivations and goals, open source and free software diverge greatly. Yet in terms of the software, the projects, and the licenses they use, they are completely synonymous. While people who identify with either group see the two movements as being at odds, the Ubuntu project sees no conflict between the two ideologies. People in the Ubuntu project identify with either group and often with both. In this book, we may switch back and forth between the terms as different projects and people in Ubuntu identify more strongly with one term or the other. For the purposes of this book, though, either term should be read as implying the other unless it is stated otherwise.

A Brief History of the Ubuntu Project

A history of Ubuntu, born in April 2004, may seem premature. However, the last six years have been full ones for Ubuntu. With its explosive growth, it is difficult even for those involved most closely with the project to track and record some of the high points. Importantly, there are some key figures whose own history must be given for a full understanding of Ubuntu. This brief summary outlines the high points of Ubuntu's history to date and gives the necessary background knowledge to understand where Ubuntu comes from.

Mark Shuttleworth

No history of Ubuntu can call itself complete without a history of Mark Shuttleworth. Shuttleworth is, undeniably, the most visible and important person in Ubuntu. More important from the point of view of history,

Shuttleworth is also the originator and initiator of the project—he made the snowball that would eventually roll on and grow to become the Ubuntu project.

Shuttleworth was born in 1973 in Welkom, Free State, in South Africa. He attended Diocesan College and obtained a business science degree in finance and information systems at the University of Cape Town. During this period, he was an avid computer hobbyist and became involved with the free and open source software community. He was at least marginally involved in both the Apache project and the Debian project and was the first person to upload the Apache Web server, perhaps the single most important piece of server software on GNU/Linux platforms, into the Debian project's archives.

Seeing an opportunity in the early days of the Web, Shuttleworth founded a certificate authority and Internet security company called Thawte in his garage. Over the course of several years, he built Thawte into the second-largest certificate authority on the Internet, trailing only the security behemoth VeriSign. Throughout this period, Thawte's products and services were built and served almost entirely from free and open source software. In December 1999, Shuttleworth sold Thawte to VeriSign for an undisclosed amount that reached into the hundreds of millions in U.S. dollars.

With his fortune made at a young age, Shuttleworth might have enjoyed a life of leisure—and probably considered it. Instead, he decided to pursue his lifelong dream of space travel. After paying approximately $20 million to the Russian space program and devoting nearly a year to preparation, including learning Russian and spending seven months training in Star City, Russia, Shuttleworth realized his dream as a civilian cosmonaut aboard the Russian Soyuz TM-34 mission. On this mission, Shuttleworth spent two days on the Soyuz rocket and eight days on the International Space Station, where he participated in experiments related to AIDS and genome research. In early May 2002, Shuttleworth returned to Earth.

In addition to space exploration and a less-impressive jaunt to Antarctica, Shuttleworth played an active role as both a philanthropist and a venture capitalist. In 2001, he founded the Shuttleworth Foundation (TSF), a non-profit organization based in South Africa. The foundation was chartered

to fund, develop, and drive social innovation in the field of education. Of course, the means by which TSF attempts to achieve these goals frequently involves free software. Through these projects, the organization has been one of the most visible proponents of free and open source software in South Africa and even the world. In the venture capital area, Shuttleworth worked to foster research, development, and entrepreneurship in South Africa with strategic injections of cash into start-ups through a new venture capital firm called HBD, an acronym for "Here Be Dragons." During this period, Shuttleworth was busy brainstorming his next big project—the project that would eventually become Ubuntu.

The Warthogs

There has been no lack of projects attempting to wrap GNU, Linux, and other pieces of free and open source software into a neat, workable, and user-friendly package. Mark Shuttleworth, like many other people, believed that the philosophical and pragmatic benefits offered by free software put it on a course for widespread success. That said, none of the offerings were particularly impressive. Something was missing from all of them. Shuttleworth saw this as an opportunity. If someone could build the great free software distribution that helped push GNU/Linux into the mainstream, he or she would come to occupy a position of strategic importance.

Shuttleworth, like many other technically inclined people, was a huge fan of the Debian project (discussed in depth later). However, many things about Debian did not fit with Shuttleworth's vision of an ideal OS. For a period of time, Shuttleworth considered the possibility of running for Debian project leader as a means of reforming the Debian project from within. With time, though, it became clear that the best way to bring GNU/Linux into the mainstream would not be from within the Debian project—which in many situations had very good reasons for being the way it was. Instead, Shuttleworth would create a new project that worked in symbiosis with Debian to build a new, better GNU/Linux system.

To kick off this project, Shuttleworth invited a dozen or so free and open source software developers he knew and respected to his flat in London in April 2004. It was in this meeting (alluded to in the first paragraphs of this

introduction) that the groundwork for the Ubuntu project was laid. By that point, many of those involved were excited about the possibility of the project. During this meeting, the members of the team—which would in time grow into the core Ubuntu team—brainstormed a large list of the things that they would want to see in their ideal OS. The list is now a familiar list of features to most Ubuntu users. Many of these traits are covered in more depth later in this introduction. The group wanted

- Predictable and frequent release cycles
- A strong focus on localization and accessibility
- A strong focus on ease of use and user-friendliness on the desktop
- A strong focus on Python as the single programming language through which the entire system could be built and expanded
- A community-driven approach that worked with existing free software projects and a method by which the groups could give back as they went along—not just at the time of release
- A new set of tools designed around the process of building distributions that allowed developers to work within an ecosystem of different projects and that allowed users to give back in whatever ways they could

There was consensus among the group that actions speak louder than words, so there were no public announcements or press releases. Instead, the group set a deadline for itself—six short months in the future. Shuttleworth agreed to finance the work and pay the developers full-time salaries to work on the project. After six months, they would both announce their project and reveal the first product of their work. They made a list of goals they wanted to achieve by the deadline, and the individuals present took on tasks. Collectively, they called themselves the Warthogs.

What Does *Ubuntu* Mean?

At this point, the Warthogs had a great team, a set of goals, and a decent idea of how to achieve most of them. The team did not, on the other hand, have a name for the project. Shuttleworth argued strongly that they should call the project Ubuntu.

Ubuntu is a concept and a term from several South African languages, including Zulu and Xhosa. It refers to a South African ideology or ethic that, while difficult to express in English, might roughly be translated as "humanity toward others," or "I am because we are." Others have described ubuntu as "the belief in a universal bond of sharing that connects all humanity." The famous South African human rights champion Archbishop Desmond Tutu explained ubuntu in this way:

> A person with ubuntu is open and available to others, affirming of others, does not feel threatened that others are able and good, for he or she has a proper self-assurance that comes from knowing that he or she belongs in a greater whole and is diminished when others are humiliated or diminished, when others are tortured or oppressed.

Ubuntu played an important role as a founding principle in postapartheid South Africa and remains a concept familiar to most South Africans today.

Shuttleworth liked the term *Ubuntu* as a name for the new project for several reasons. First, it is a South African concept. While the majority of the people who work on Ubuntu are not from South Africa, the roots of the project are, and Shuttleworth wanted to choose a name that represented this. Second, the project emphasizes the definition of individuality in terms of relationships with others and provides a profound type of community and sharing—exactly the attitudes of sharing, community, and collaboration that are at the core of free software. The term represented the side of free software that the team wanted to share with the world. Third, the idea of personal relationships built on mutual respect and connections describes the fundamental ground rules for the highly functional community that the Ubuntu team wanted to build. *Ubuntu* was a term that encapsulated where the project came from, where the project was going, and how the project planned to get there. The name was perfect. It stuck.

Creating Canonical

In order to pay developers to work on Ubuntu full-time, Shuttleworth needed a company to employ them. He wanted to pick some of the best people for the jobs from within the global free software and open source communities. These communities, inconveniently for Shuttleworth, know no national and geographic boundaries. Rather than move everyone to a

single locale and office, Shuttleworth made the decision to employ these developers through a virtual company. While this had obvious drawbacks in the form of high-latency and low-bandwidth connections, different time zones, and much more, it also introduced some major benefits in the particular context of the project. On one hand, the distributed nature of employees meant that the new company could hire individuals without requiring them to pack up their lives and move to a new country. More important, it meant that *everyone* in the company was dependent on IRC, mailing lists, and online communication mechanisms to do their work. This unintentionally and automatically solved the water-cooler problem that plagued many other corporately funded free software projects—namely, that developers would casually speak about their work in person and cut the community and anyone else who didn't work in the office out of the conversation completely. For the first year, the closest thing that Canonical had to an office was Shuttleworth's flat in London. While the company has grown and now has several offices around the world, it remains distributed, and a large number of the engineers work from home. The group remains highly dependent on Internet collaboration.

With time, the company was named Canonical. The name was a nod to the project's optimistic goals of becoming the canonical place for services and support for free and open source software and for Ubuntu in particular. *Canonical*, of course, refers to something that is accepted as authoritative. It is a common word in the computer programmer lexicon. It's important to note that being canonical is like being standard; it is not coercive. Unlike holding a monopoly, becoming the canonical location for something implies a similar sort of success—but *never* one that cannot be undone and never one that is exclusive. Other companies will support Ubuntu and build operating systems based on it, but as long as Canonical is doing a good job, its role will remain central.

The Ubuntu Community

By now you may have noticed a theme that permeates the Ubuntu project on several levels. The history of free software and open source is one of a profoundly effective *community*. Similarly, in building a GNU/Linux distribution, the Ubuntu community has tried to focus on an ecosystem model—an organization of organizations—in other words, a community.

Even the definition of the term *ubuntu* is one that revolves around people interacting in a community.

It comes as no surprise, then, that an "internal" community plays heavily into the way that the Ubuntu distribution is created. While the Ubuntu 4.10 version (Warty Warthog) was primarily built by a small number of people, Ubuntu achieved widespread success only through contributions by a much larger group that included programmers, documentation writers, volunteer support staff, and users. While Canonical employs a core group of several dozen active contributors to Ubuntu, the distribution has, from day one, encouraged and incorporated contributions from *anyone* in the community and rewards and recognizes contributions by all. Rather than taking center stage, paid contributors are *not* employed by the Ubuntu project—instead they are employed by Canonical, Ltd. These employees are treated simply as another set of community members. They must apply for membership in the Ubuntu community and have their contributions recognized in the same way as anyone else. All non-business-related communication about the Ubuntu project occurs in public and in the community. Volunteer community members occupy a majority of the seats on the two most important governing boards of the Ubuntu project: the Technical Board, which oversees all technical matters, and the Community Council, which approves new Ubuntu members and resolves disputes. Seats on both boards are approved by the relevant community groups, developers for the Technical Board and Ubuntu members for the Community Council.

In order to harness and encourage the contributions of its community, Ubuntu has striven to balance the important role that Canonical plays with the value of empowering individuals in the community. The Ubuntu project is based on a fundamental belief that great software is built, supported, and maintained only in a strong relationship with the individuals who use the software. In this way, by fostering and supporting a vibrant community, Ubuntu can achieve much more than it could through paid development alone. The people on the project believe that while the contributions of Canonical and Mark Shuttleworth have provided an important catalyst for the processes that have built Ubuntu, it is the community that has brought the distribution its success to date. The project members believe that it is only through increasing reliance on the community that

the project's success will continue to grow. The Ubuntu community won't outspend the proprietary software industry, but it is very much more than Microsoft and its allies can afford.

Finally, it is worth noting that, while this book is official, neither of the authors is a Canonical employee. This book, like much of the rest of Ubuntu, is purely a product of the project's community.

Ubuntu Promises and Goals

So far, this Introduction has been about the prehistory, history, and context of the Ubuntu project. After this introduction, the book focuses on the distribution itself. Before proceeding, it's important to understand the goals that motivated the project.

Philosophical Goals

The most important goals of the Ubuntu project are philosophical in nature. The Ubuntu project lays out its philosophy in a series of documents on its Web site. In the most central of these documents, the team summarizes the charter and the major philosophical goals and underpinnings:

> Ubuntu is a community-driven project to create an operating system and a full set of applications using free and Open Source software. At the core of the Ubuntu Philosophy of Software Freedom are these core philosophical ideals:
>
> 1. Every computer user should have the freedom to run, copy, distribute, study, share, change, and improve their software for any purpose without paying licensing fees.
> 2. Every computer user should be able to use their software in the language of their choice.
> 3. Every computer user should be given every opportunity to use software, even if they work under a disability.

The first item should be familiar by now. It is merely a recapitulation of Stallman's free software definition quoted earlier in the section on free software history. In it, the Ubuntu project makes explicit its goal that every user of software should have the freedoms required by free software. This

is important for a number of reasons. First, it offers users all of the practical benefits of software that runs better, faster, and more flexibly. More important, it gives every user the capability to transcend his or her role as a user and a consumer of software. Ubuntu wants software to be empowering and to work in the ways that users want it to work. Ubuntu wants all users to have the ability to make sure it works for them. To do this, software *must* be free, so Ubuntu makes this a requirement and a philosophical promise.

Of course, the core goals of Ubuntu do not end with the free software definition. Instead, the project articulates two new, but equally important, goals. The first of these, that all computer users should be able to use their computers in their chosen languages, is a nod to the fact that the majority of the world's population does not speak English while the vast majority of software interacts only in that language. To be useful, source code comments, programming languages, documentation, and the texts and menus in computer programs must be written in *some* language. Arguably, the world's most international language is a reasonably good choice. However, there is no language that everyone speaks, and English is not useful to the majority of the world's population that does not speak it. A computer can be a great tool for empowerment and education, but only if the user can understand the words in the computer's interface. As a result, Ubuntu believes that it is the project's—and community's—responsibility to ensure that *every* user can easily use Ubuntu to read and write in the language with which he or she is most comfortable.

Finally, just as no person should be blocked from using a computer simply because he or she does not know a particular language, no user should be blocked from using a computer because of a disability. Ubuntu must be accessible to users with motor disabilities, vision disabilities, and hearing disabilities. It should provide input and output in a variety of forms to account for each of these situations and for others. A significant percentage of the world's most intelligent and creative individuals has disabilities. Ubuntu's impact should not be limited to any subset of the world when it can be fully inclusive. More important, Ubuntu should be able to harness the ability of these individuals as community members to build a better and more effective community.

Conduct Goals and Code of Conduct

If Ubuntu's philosophical commitments describe the *why* of the Ubuntu project, the Code of Conduct (CoC) describes Ubuntu's *how*. Ubuntu's CoC is, arguably, the most important document in the day-to-day operation of the Ubuntu community and sets the ground rules for work and cooperation within the project. Explicit agreement to the document is the only criterion for becoming an officially recognized Ubuntu activist—an Ubuntero—and is an essential step toward membership in the project.

The CoC covers "behavior as a member of the Ubuntu Community, in any forum, mailing list, wiki, Web site, IRC channel, install-fest, public meeting, or private correspondence." The CoC goes into some degree of depth on a series of points that fall under the following headings:

- Be considerate.
- Be respectful.
- Be collaborative.
- When you disagree, consult others.
- When you are unsure, ask for help.
- Step down considerately.

Many of these headings seem like common sense or common courtesy to many, and that is by design. Nothing in the CoC is controversial or radical, and it was never designed to be.

More difficult is that nothing is easy to enforce or decide because acting considerately, respectfully, and collaboratively is often very subjective. There is room for honest disagreements and hurt feelings. These are accepted shortcomings. The CoC was not designed to be a law with explicit prohibitions on phrases, language, or actions. Instead, it aims to provide a constitution and a reminder that considerate and respectful discussion is *essential* to the health and vitality of the project. In situations where there is a serious disagreement on whether a community member has violated or is violating the code, the Community Council is available to arbitrate disputes and decide what action, if any, is appropriate.

Nobody involved in the Ubuntu project, including Mark Shuttleworth and the other members of the Community Council, is above the CoC. The CoC is never optional and never waived. In fact, the Ubuntu community recently created a Leadership Code of Conduct (LCoC), which extends and expands on the CoC and describes additional requirements and expectations for those in leadership positions in the community. Of course, in no way was either code designed to eliminate conflict or disagreement. Arguments are at least as common in Ubuntu as they are in other projects and online communities. However, there is a common understanding within the project that arguments should happen in an environment of collaboration and mutual respect. This allows for *better* arguments with *better* results—and with less hurt feelings and fewer bruised egos.

While they are sometimes incorrectly used as such, the CoC and LCoC are not sticks to be wielded against an opponent in an argument. Instead, they are useful points of reference upon which consensus can be assumed within the Ubuntu community. Frequently, if a group in the community feels a member is acting in a way that is out of line with the code, the group will gently remind the community member, often privately, that the CoC is in effect. In almost all situations, this is enough to avoid any further action or conflict. Very few CoC violations are ever brought before the Community Council.

Technical Goals

While a respectful community and adherence to a set of philosophical goals provide an important frame in which the Ubuntu project works, Ubuntu is, at the end of the day, a technical project. As a result, it only makes sense that in addition to philosophical goals and a project constitution, Ubuntu also has a set of technical goals.

The first technical goal of the project, and perhaps the most important one, is the coordination of regular and predictable releases—something particularly important to server users. In April 2004, at the Warthogs meeting, the project set a goal for its initial proof-of-concept release six months out. In part due to the resounding success of that project, and in larger part due to the GNOME release schedule, the team has stuck to a regular and predictable six-month release cycle and has only once chosen

to extend the release schedule by six weeks and only after obtaining community consensus on the decision. The team then redoubled its efforts and made the next release in a mere four and a half months, putting its release schedule back on track. Frequent releases are important because users can then use the latest and greatest free software available—something that is essential in a development environment as vibrant and rapidly changing and improving as the free software community. Predictable releases are important, especially to businesses, because predictability means that they can organize their business plans around Ubuntu. Through consistent releases, Ubuntu can provide a platform upon which businesses and derivative distributions can rely to grow and build.

While releasing frequently and reliably is important, the released software must then be supported. Ubuntu, like all distributions, must deal with the fact that *all* software has bugs. Most bugs are minor, but fixing them may introduce even worse issues. Therefore, fixing bugs after a release must be done carefully or not at all. The Ubuntu project engages in major changes, including bug fixes, *between* releases only when the changes can be extensively tested. However, some bugs risk the loss of users' information or pose a serious security vulnerability. These bugs are fixed immediately and made available as updates for the released distribution. The Ubuntu community works hard to find and minimize all types of bugs before releases and is largely successful in squashing the worst. However, because there is always the possibility that more of these bugs will be found, Ubuntu commits to supporting *every* release for 18 months after it is released. In the case of Ubuntu 6.06 LTS (Dapper Drake), released in 2006, the project went well beyond even this and committed to support the release for three full years on desktop computers and for five years in a server configuration (LTS stands for LongTerm Support). This proved so popular with businesses, institutions, and the users of Ubuntu servers that Ubuntu 8.04 (Hardy Heron) was named as Ubuntu's second LTS release with similar three- and five-year desktop and server extended support commitments. These five-year support commitments are specifically designed for server users and make Ubuntu a much more attractive option for an important class of server users.

This bipartite approach to servers and desktops implies the third major technical commitment of the Ubuntu project and, in a sense, the most

important for this book: support for both servers and desktop computers in separate but equally emphasized modes. While Ubuntu continues to be more well known, and perhaps more popular, in desktop configurations, there exist teams of Ubuntu developers focused on both server and desktop users. The Ubuntu project believes that both desktops and servers are essential and provides installation methods on every CD for both types of systems. Ubuntu provides tested and supported software appropriate to the most common actions in both environments and documentation for each. LTS releases in particular mark an important step toward catering to users on the server.

Finally, the Ubuntu project is committed to making it as easy as possible for users to transcend their roles as consumers and users of software and to take advantage of each of the freedoms central to the Ubuntu philosophy. As a result, Ubuntu has tried to focus its development around the use and promotion of a single programming language, Python. The project has worked to ensure that Python is widely used throughout the system. By ensuring that many applications and many of the "guts" of the system are written in or extensible in Python, Ubuntu is working to ensure that users need to learn only one language in order to take advantage of, automate, and tweak many parts of their systems.

Canonical and the Ubuntu Foundation

While Ubuntu is driven by a community, several groups play an important role in its structure and organization. Foremost among these are Canonical, Ltd., a for-profit company introduced as part of the Ubuntu history description, and the Ubuntu Foundation, which is introduced later in this section.

Canonical, Ltd.

As mentioned earlier, Canonical, Ltd., is a company founded by Mark Shuttleworth with the primary goal of developing and supporting the Ubuntu distribution. Many of the core developers on Ubuntu—although no longer a majority of them—work full-time or part-time under contract for Canonical, Ltd. This funding by Canonical allows Ubuntu to make the type of support commitments that it does. Ubuntu can claim that it will

release in six months because releasing, in one form or another, is something that the paid workers at Canonical can ensure. As an all-volunteer organization, Debian suffered from an inability to set and meet deadlines—volunteers become busy or have other deadlines in their paying jobs that take precedence. By offering paying jobs to a subset of developers, Canonical can set support and release deadlines and ensure that they are met.

In this way, Canonical ensures that Ubuntu's bottom-line commitments are kept. Of course, Canonical does not fund all Ubuntu work, nor could it. Canonical can release a distribution every six months, but that distribution will be made *much* better and more usable through contributions from the community of users. Most features, most new pieces of software, almost all translations, almost all documentation, and much more are created outside of Canonical. Instead, Canonical ensures that deadlines are met and that the essential work, regardless of whether it's fun, gets done.

Canonical, Ltd., was incorporated on the Isle of Man—a tiny island nation between Wales and Ireland that is mostly known as a haven for international businesses. Since Canonical's staff is sprinkled across the globe and no proper office is necessary, the Isle of Man seemed as good a place as any for the company to hang its sign.

Canonical's Service and Support

While it is surprising to many users, fewer than half of Canonical's employees work on the Ubuntu project. The rest of the employees fall into several categories: business development, support and administration, and development on the Bazaar and Launchpad projects.

Individuals involved in business development help create strategic deals and certification programs with other companies—primarily around Ubuntu. In large part, these are things that the community is either ill suited for or uninterested in as a whole. One example of business development work is the process of working with companies to ensure that their software (usually proprietary) is built and certified to run on Ubuntu. For example, Canonical worked with IBM to ensure that its popular DB2 database would run on Ubuntu and, when this was achieved, worked to have

Ubuntu certified as a platform that would run DB2. Similarly, Canonical worked with Dell to ensure that Ubuntu could be installed and supported on Dell laptops as an option for its customers. A third example is the production of this book, which, published by Pearson Education's Prentice Hall imprint, was a product of work with Canonical.

Canonical also plays an important support role in the Ubuntu project in three ways. First, Canonical supports the development of the Ubuntu project. For example, Canonical system administrators ensure that the servers that support development and distribution of Ubuntu are running. Second, Canonical helps Ubuntu users and businesses directly by offering phone and e-mail support. Additionally, Canonical has helped build a large commercial Ubuntu support operation by arranging for support contracts with larger companies and organizations. This support is over and above the free (i.e., gratis) support offered by the community—this commercial support is offered at a fee and is either part of a longer-term flat-fee support contract or is pay-per-instance. By offering commercial support for Ubuntu in a variety of ways, Canonical aims to make a business for itself and to help make Ubuntu a more palatable option for the businesses, large and small, that are looking for an enterprise or enterprise-class GNU/Linux product with support contracts like those offered by other commercial GNU/Linux distributions.

Finally, Ubuntu supports other support organizations. Canonical does not seek or try to enforce a monopoly on Ubuntu support; it proudly lists *hundreds* of other organizations offering support for Ubuntu on the Ubuntu Web pages. Instead, Canonical offers what is called second-tier support to these organizations. Because Canonical employs many of the core Ubuntu developers, the company is very well suited to taking action on many of the tougher problems that these support organizations may run into. With its concentrated expertise, Canonical can offer this type of backup, or secondary, support to these organizations.

The Ubuntu Foundation

Finally, in addition to Canonical and the full Ubuntu community, the Ubuntu project is supported by the Ubuntu Foundation, which was announced by Shuttleworth with an initial funding commitment of $10 million. The foun-

dation, like Canonical, is based on the Isle of Man. The organization is advised by the Ubuntu Community Council.

Unlike Canonical, the foundation does not play an active role in the day-to-day life of Ubuntu. At the moment, the foundation is little more than a pile of money that exists to endow and ensure Ubuntu's future. Because Canonical is a young company, many companies and individuals find it difficult to trust that Canonical will be able to provide support for Ubuntu in the time frames (e.g., three to five years) that it claims it will be able to. The Ubuntu Foundation exists to allay those fears.

If something bad were to happen to Shuttleworth or to Canonical that caused either to be unable to support Ubuntu development and maintain the distribution, the Ubuntu Foundation exists to carry on many of Canonical's core activities well into the future. Through the existence of the foundation, the Ubuntu project can make the types of long-term commitments and promises it does.

The one activity in which the foundation can and does engage is receiving donations on behalf of the Ubuntu project. These donations, and only these donations, are then put to use on behalf of Ubuntu in accordance with the wishes of the development team and the Technical Board. For the most part, these contributions are spent on "bounties" given to community members who have achieved important feature goals for the Ubuntu project.

History of Ubuntu Server

The first "production" machines to run Ubuntu were Canonical's own development machines in its data center in London. In this sense, Ubuntu has been used on servers since day one, and Ubuntu has *always* been a server operating system. Of course, as we hinted in the welcome at the beginning of this Introduction, this has not always been universally recognized. After the first release, public perception was tilted so far toward the idea of Ubuntu as a desktop release that when the developers convened the first of their biannual developer summits after the first full release cycle, one of the most important items on the agenda was thinking about Ubuntu on servers and how to support it.

The Ubuntu Server project, as a result, was at least as much a marketing project as it was a technical project. Sure, there were ways that the team could make Ubuntu better for servers—and they spent plenty of time working and thinking about that—but the biggest problem they faced was simply communicating the message that Ubuntu already was great for servers to all their users and potential users.

Eventually Canonical funded the creation of a graphical installer, but in the first few releases there was just a single, nongraphical installer based on Debian's very descriptively named Debian Installer project. In the initial Ubuntu release, a user installing Ubuntu was given a choice between two modes: "Desktop"—which was self-explanatory enough—and "Custom." Custom, in the minds of the early developers, was what anyone would want for a server. Custom installed just the bare minimum set of packages and then put the users into this base install and prompted them to install the packages that they wanted on their system. It provided users with the bare-bones system and encouraged them to customize it. The first action of the Ubuntu Server project was purely superficial: The "Custom" install was renamed "Server." Although no code had changed, Ubuntu Server almost immediately began getting more recognition. If one had to pick a single point in time that the Ubuntu Server project was born, it would be this moment.

Ubuntu Server isn't actually any different from other flavors of Ubuntu. As the desktop has moved on to a new graphical installer based on a live CD, Ubuntu Server has its own installer that gives users access to features like RAID and LVM that are much more interesting to server users. Certainly, there are some pieces of software that are likely to end up on servers and unlikely to end up on desktops—things like Web servers and mail servers. When we say that the server edition will be supported, we mean these applications plus the core, so it certainly seems most accurate to refer to these as being within the purview of Ubuntu Server.

But at the end of the day, the server and desktop packages come out of a single repository. This fact, plus the integration among the teams of people working on different parts of the project—most core developers work on bits and pieces that get used and reused in server, desktop, and other editions—introduces a fuzziness that makes it hard to pin down just what

Ubuntu Server *is*. Of course, it also means that Ubuntu Server gets to benefit from the work, bug reporting, and bug fixing in those core parts of the operating system that every Ubuntu user shares.

Ubuntu Server now can roughly be interpreted to refer to a collection of resources that are particularly aimed at and used by server users. Most obviously, it involves the custom install discs that you'll be using when you install Ubuntu Server on your machine. It also refers to the collections of supported software that are installed primarily on servers—most of the software that the rest of this book will discuss in more detail. It also refers to a mass of documentation, to which this book represents the latest addition, that provides answers to questions. In a broader sense, certifications of software and training programs for administrators occupy another point in the growing Ubuntu Server constellation.

But most of all, and in the Ubuntu tradition, Ubuntu Server refers to a community. It's a community of developers who use Ubuntu on servers, who care deeply about Ubuntu on servers, and who work tirelessly to make sure that Ubuntu performs as well as possible on servers everywhere. Of course, Ubuntu Server also refers to the growing community of people who are primarily not contributing through code but who are at least as important. These people spend time in the support of IRC channels, send e-mail to the mailing lists, and post in the forums. These users help other users, file bugs, may contribute their own fixes to documentation, and contribute in myriad ways and in a variety of venues.

When you "graduate" beyond what this book can teach you, Ubuntu represents those people who will help you take your next steps. They are the people described in more depth in the server resources chapter (Chapter 13) of this book. This is the group you will join when you participate in the Ubuntu project. Let us be the first to welcome to you to the Ubuntu Server community.

Simple, Secure, Supported

Early on, the initial core Ubuntu team—of which one of this book's authors was lucky enough to be a part—resisted the idea of the server version of Ubuntu. Or rather, they resisted the idea of a server distribution in

the way that other GNU/Linux distributions had produced them and the way in which they were commonly thought of. The team was more than happy with running Ubuntu on servers, of course, but they resisted the idea of "server distributions" because of the way that Red Hat, SuSE, and the other big distributions built their businesses around "enterprise Linux" distributions that were big, clunky, and expensive. The result was, in the eyes of many of the early Ubuntu core developers and Canonical employees, top-heavy monstrosities. That's not what Ubuntu is about.

The big server-based GNU/Linux distributions seemed to be competing over who included more services, more features, and more bells and whistles. Distribution 1 would have a Web server, an FTP server, a DNS server, several file servers, and a mail server. Distribution 2 would have all of those plus a DHCP server! A brand-new install of one of these "server distributions" would be running dozens of daemons—each taking up many megabytes of memory, loads of disk space, and (most important) lots of administrator time when they failed or interfered with something else. But worst of all, most of these daemons lay completely unused on most installs.

And if that wasn't enough, the server installs would then run firewalls to keep people from accessing all these now-open services and to prevent users from exposing security vulnerabilities from their newly installed machines. Of course, there would be regular upgrades, security releases, and the like, to update all these now-firewalled services that nobody was using. Debian provided one alternative model that focused on custom installations of just what people needed. Among an elite group of sysadmins in the late 1990s and the early 2000s, Debian had become the server OS of choice. Because nearly everyone on the early Ubuntu team was a Debian developer, it was to this model and to Debian technology that the Ubuntu team first turned.

Of course, the commercial GNU/Linux server market was not all horrible. For example, the early Ubuntu developers liked the idea of commercial support for its servers. They liked the idea of regular, predictable releases. As Debian developers, they all knew someone who wanted to install a simple, custom version of Debian on a server but who, because of the lack of commercial support and accountability, had been rejected by a higher-

up in the company or organization. They liked the idea of a company using Debian's technology to offer simple, custom server installs but could offer a commercial support contract. The Warthogs, and lots of folks like them, had waited years for this, but nobody had stepped up to the plate.

As we described in the previous section, an Ubuntu server install was simply a bare-bones installation. We were all administrators—at least of our own machines—and when *we* installed servers, we started out with "naked" machines so that we could choose every application, every daemon, every service that would go onto the machine. As administrators, we wanted the *options* of the big enterprise distributions, but we wanted to be able to choose those options ourselves. Like all administrators, we used servers to solve problems and to offer services to our users. These problems and needs are unique and, as a result, the cookie-cutter model of GNU/Linux servers was always a poor match.

And so that is what the Warthogs built and it is what Ubuntu Server remains today. At first, some people were confused. Ubuntu's server offering was panned in several reviews for not including a firewall by default. But Ubuntu installed *no open ports by default*, so there was nothing to firewall! Of course, Ubuntu provided several firewalls for users to install *if they wanted one*, but Ubuntu left the decision to install a firewall, just like the decision to install services that might require one, up to the server's administrator. For all installations but for server installations in particular, Ubuntu's goal is to make the default installation simple and secure and to put the user in the driver's seat. Ubuntu's job, as distribution producer, is to make it as close to drop-dead simple for system administrators to do their jobs. In an Ubuntu Server install, every machine is exactly as complicated as the administrator has requested but never any more than necessary. No extra services or unnecessary features are included—although they are waiting in the wings for when they become necessary and are easily installable in ways that are described in Chapter 3.

One important effect of this simplicity is security. When there is less going on, there is simply less to go wrong. But, of course, the Ubuntu team has taken this many steps further and pursued proactive security in a number of other contexts. Ubuntu's first release was held up for one day because a single open port was found in the default release. The goal of a machine with

no open ports by default was more important than an on-time release. Ubuntu's CTO and the chairman of the Ubuntu technical board, Matt Zimmerman, is a longtime security-focused developer who made nearly all of Debian's security updates for more than a year before joining the Warthogs. As Ubuntu struggles over hard decisions about what to include or to pass up for inclusion in the distribution, the most important questions continue to be ones of security and support. "Can we—and we do want to—maintain security support and provide security releases for this software for the next 18 months?" Every piece of software included by default is subjected to this question, and many popular pieces of software are kept out because Ubuntu is reluctant to support them. Inclusion as an officially supported package means that a server admin can trust the software—both because Canonical has indicated that it trusts it and because Canonical has promised to clean up any security messes that occur through fixing important bugs and issuing a fixed package. Canonical's security guarantee goes beyond security bugs to other bugs that might result in data loss. While there are no guarantees beyond this, Canonical makes many dozens of new updates per release that fix other important bugs in the distribution as well. The result is a rock-solid system with a commitment to continue.

With customizability, security, and support, Ubuntu truly is ready for the data room. The rest of this book will show you how.

CHAPTER 3

Package Management

THIS CHAPTER BEGINS WITH a discussion of packages in general while focusing on the core features of packages and package management systems that cross most GNU/Linux distributions. In this discussion, I explain what packages are and what a package management system does. While I turn to examples from Ubuntu throughout, this discussion focuses on building a strong conceptual grounding. After establishing a solid grounding, I introduce Debian packages—the types of packages that Ubuntu uses—and give a brief view of the very different types of packages: source packages and binary packages. Most of the rest of the chapter focuses on package management in Ubuntu using the command-line tools. While many users of Ubuntu on the desktop are familiar with updating their system, this chapter focuses on the way this is done without a desktop system. It covers the basics and works up to some more advanced uses of a packaging system that many server administrators find useful. Finally, I touch on the process through which advanced users and administrators can create, modify, and redistribute their own packages.

Introduction to Package Management

On Ubuntu—and in other GNU/Linux environments—packages are the primary way that software is built, deployed, and installed. Nearly every major GNU/Linux operating system distributes software, both binary software and source code, in packages. These packages are usually either in the Rpm package format (RPM) or in the Debian package format (DEB) for binary software or in corresponding "source" RPM and DEB formats. With its close relationship to the Debian project as a project that continues to be based on Debian's work, Ubuntu naturally uses DEB format packages.

Very simply, packages are an alternative to downloading, building, and installing software from scratch. They offer a host of advantages in terms of installation, removal, monitoring, and handling interactions between pieces of software over the standard "build from source" model. Since packaging is not common outside of the GNU/Linux world—or at least not described in the same terms—it is worth going into some background on packaging before I describe how it is done on Ubuntu systems.

Background on Packages

Nearly every GNU/Linux-based operating system—Fedora, RHEL, openSUSE, Slackware, Debian, and others—includes an almost entirely overlapping core selection of software. By definition, each of these OSes includes Linus Torvald's Linux kernel and a large chunk of the GNU project's developer- and user-oriented applications that are necessary to build and use it. Most also include server-oriented software like OpenSSH and Apache, either the XFree86 or X.Org implementation of the X Windowing System, and what is often an extremely expansive collection of both command-line and graphically based applications. Although people often throw the term around, it is important to establish that this collection of software is collectively referred to as a distribution. Ubuntu is a distribution. When people refer to "Linux" as an operating system, they are usually referring to a Linux or GNU/Linux distribution.

The primary goal of all distributions is the automatic installation, configuration, removal, maintenance, and update of software—both through the creation of infrastructure for this purpose and in the creation of modified versions of the preexistent software. The latter customization of existing software in this specialized way is the act of "packaging," and it constitutes the vast proportion of work done by Ubuntu developers. It constitutes, to a large degree, what Ubuntu is over and above the software that Ubuntu includes. And while packaging is primarily the work of distribution makers like Ubuntu, it can also be done by both the users of distributions, for the clean integration of "unpackaged" pieces of software into their systems, and by software vendors who wish to allow for easier installation and maintenance of software by their users.

What Are Packages?

The creation of a package—on Ubuntu or elsewhere—begins with the software in need of being packaged. In most, but not all, cases, this involves the procurement of source code. In all situations, it involves code from an original source, usually referred to in the distribution world as an "upstream"

source. The packager's first addition to the code here will be the creation of extra metadata, which usually includes

- The name of the software
- The name of the upstream author and the person creating the package
- The license of the software
- The upstream location of the software (or a description of where it was obtained)
- The architecture or architectures on which the software is guaranteed to run
- Information for classifying the software that often has to do with the use of the package, primarily to help people who are browsing for packages
- A description of the software in a computer-parsable format
- Information on the importance or "priority" of the package within the larger Ubuntu system (e.g., essential, optional)

This information will be used by either a packaging system or a series of package selection tools to allow users to search, sort, query, or interact with installed or available software—one of the package system's jobs. However, while this type of metadata is important in that it allows users to find (and find out about) their software, by far the most important group of metadata added to a package relates to the documentation of the relationship of the software in the package to software in other packages within the distribution. While the syntax and semantics of this vary widely between distributions, they include relationships to

- Other software that the software requires to be built
- Other software that the software requires to be installed or configured
- Other software that the software requires to be run
- Other software with which the software cannot be installed or used simultaneously

- Other software for which the software can be used as a drop-in replacement
- Other software that can enhance or improve the software

Modern package systems record even more information. For example, configuration files, unlike normal files, cannot always be simply replaced with a new version when the software is upgraded. As a result, packaging systems have grown to include several pieces of infrastructure for querying users and for maintaining core configuration information over time and across upgrades of the package that requires changes to configuration files. Finally, a more recently realized goal of packages is to provide a structure around which package metadata—such as descriptions—can be translated to provide users with an interface to software localized to their language, script, and culture. Details on accessing and creating all of this metadata in Ubuntu packages are included in the subsequent sections.

Basic Functions of Package Management

A wide range of functionality can be considered core functions of package management systems. The functions are usually implemented by a low-level tool or suite of tools. This script is `dpkg` and associated scripts in the case of Ubuntu and Debian. These tools were, until several years ago, the primary way that most users manipulated packages, but with the creation of higher-level package management tools that provide "front ends" to these tools, most users of package-based systems rarely use them, although they are still highly central for developers or system administrators who build their own packages. Broadly and somewhat imprecisely, many of these tools are referred to as APT on Debian and Ubuntu.

The first goal of packaging is automating the compilation of software. DEB-format packages provide two formats: one for source packages and one for binary packages. These source packages are an excellent system for the distribution and compilation of source code. Packages are, in Ubuntu and elsewhere, designed to be built noninteractively and—in the case of official Ubuntu packages—can be built automatically on a range of different architectures by automatic package-building software called "autobuilders."

Packages provide a simple—usually one command—method for building that is consistent across all packages. Issues of build configurations and choices are addressed ahead of time by the packager. The cost is build-time configurability, but the payoffs, as you will see in the rest of the chapter, are huge. Necessary build-time dependencies are declared in the packages so that these can be satisfied automatically. For example, architecture-dependent source packages (i.e., packages that must be rebuilt for each architecture) are uploaded to Ubuntu as source and are, in most cases, automatically built on all architectures supported by Ubuntu without any changes to the source package.

Any number of binary packages can be created from a single source package. The creation of multiple binary packages from a single source package can be useful for large projects that release large or monolithic source packages containing a wide variety of different pieces of software—or even highly related pieces of software and/or documentation that it may be advantageous to split. An example of the former case is the XFree86 windowing system—now replaced by the already modularized X.Org—which was contained in one source package but would create upward of several dozen binary packages. Packaging, in this case, is what allowed users to distribute, install, and remove the Xserver independently from the terminal emulator, xlib library package, or window manager.

As can be inferred from the preceding discussion, a key benefit of packaging systems is that they help automate the installation of software. When a binary package is installed:

- The "contents" of the software can be verified to assure integrity of the package. The origin of the software can be verified using cryptographic authentication.
- The dependencies of the software can be analyzed and the system can be queried on the installation state of the software on which the software being installed depends. If the dependencies are unsatisfied, the user is prompted as to the lack and the nature of the required software, and the installation is aborted.
- The user installing the package can be queried for configuration options at some point during the installation process. Answers to

these queries can be saved on the system and then used in the customization of a configuration file for the software being installed.

- The contents of the package are stored on the system.
- Metadata and accounting information of a variety of forms are placed in a per-system database to include both current information on the packages installed and their state of installation (e.g., installed but unconfigured), the list of files and to which package they belong, and other information.

Perhaps the most central element here is the check against dependencies of the package being installed and the list of packages already installed on the system. With information on dependencies, users can, at a glance, determine which software is required to run the software in the package. As a result, people writing software that will ultimately be packaged can easily write for and deploy software built against shared libraries. The success of package systems is one reason for the wide use of dynamically linked shared libraries in the GNU/Linux environment.

When a user wants to remove a piece of software, the packaging system, with its catalog of the files belonging to the package and the actions done during installation, is well suited to help ensure a clean uninstallation as well.

While similar to installation, the automatic upgrade of software is another area where the package system can be employed with similarly useful results: Users of package systems can safely and easily upgrade from one version of a piece of software to another. The upgrade of the software will work almost identically to the installation of the software. In most cases, software is installed on top of the existing package, and files that are no longer provided by the package are removed. Configuration files that were customized by the installation and have not since been changed by the user can be automatically regenerated by the user, or the user can be prompted to view and merge changes.

Dependency information can play an important role in the upgrade of packages involving shared libraries. In the case of ABI changes, a packaging system will alert users that an upgrade of a package cannot be completed without the installation of a new library, and users can also be alerted to

other packages that will break in this upload. As a result, users can structure uploads—or the system can structure it for them—so that API and ABI breakage is not unanticipated, and users can ensure that all packages that depend on a single shared library can be upgraded in tandem.

Finally, at any point, users can use the cryptographic signature on a package and the list of hashes (usually MD5 sums) of the files included in that package to verify the integrity of the files on their system against corruption or compromise by an attacker.

Advanced Functions of Package Management Systems

While these features lead to the powerful potential to manage software on a system, packaging systems with *only* these features—essentially, the state of packaging in the mid-1990s—introduced important limitations. Large-scale API and ABI transitions required downloading many packages and a high degree of coordination by the user. Users were forced to figure out the dependency status of programs during an installation or upgrade and then find, download, and do simultaneous installations of new pieces of software. For complex pieces of software with many dependencies, this process was often exceedingly tedious.

As a result, most system upgrades and ABI/API changes were done with large upgrade scripts between releases of a distribution. Users would be expected to install every package involved in a major transition at once with an upgrade script that would structure the order correctly and handle dependencies appropriately. While these problems are limitations of a limited package management system, they are mostly problems that exist outside of package management systems. Without a package management system, shared libraries that undergo API and ABI changes are either never or rarely approached (with dangerous consistency and security implications to each) or are subject to the same limitations without the warnings that a packaging system provides.

Spurred on by the Debian project's creation of a program called `dselect` and its frequently lauded Advanced Package Tools (APT, originally named *deity* and implemented primarily in a program called `apt-get`), the last half-decade has seen a major evolution in the scope and success of package managers. Most of these tools are levels of abstraction upon or "front

ends" to the lower-level package management tools previously described. Like most other DEB-based distributions, Ubuntu uses `apt-get`, Aptitude, `dselect`, and the graphical front end Synaptic.

As the ability to track and catalog dependencies is perhaps the single most important aspect of any package management system, the primary function of these advanced tools has been to add classes of functionality on top of the extant package tools and to operate on packages in a more-than-one-at-a-time manner. Each of these tools contains additional databases that describe not only the packages installed but also the packages that are *available* as candidates for installation through package archives stored locally, on CD, or (in almost all situations today) over a network.

These systems can automatically sort out dependencies and orders, download packages (including dependencies), install the dependencies first, and then install and configure the package in question using the lower-level tools detailed in the previous section.

Similarly, the same advanced tools can be used to uninstall packages. If, for example, a user wants to uninstall a shared library, he or she is prompted with a screen that describes the consequences as a list of packages that must be uninstalled because their dependencies will no longer exist on the system after the uninstallation. Upgrades that involve changing dependencies (e.g., replaced packages) can also be handled through this system.

The real possibilities of such systems are visible when the dependency aspects of a package change over time or when multiple packages can act as drop-in replacements. A package that requires the ability to send mail can depend only on a virtual package "provided" by other packages. New versions of packages can conflict with and declare that they "replace" other packages or provide the functionality of the original package. If, for example, multiple packages are merged into a single package that obsoletes the three other packages, an advanced package system should be able to track the changing dependency information and make the correct decision during upgrade. Along these lines, most advanced package management tools give users the ability to do strategic "smart upgrades" of every package on the system to the newest version of the packages available using the data declared in the package dependencies.

Even more exciting for some users, it is possible to track an in-development version of a GNU/Linux operating system and upgrade every day to the latest version of everything. The package manager can figure out safe upgrade paths and take it from there. During these upgrades, ABI and API version changes can also be automatically handled because the system will refuse to do a full upgrade of a library until all of the packages installed on the system that depend on the package with the shared library can be upgraded at once. The system will not need to keep or track multiple versions of a shared library over time.

Debian Packages

As was mentioned earlier in this book, the Ubuntu project is based on the Debian GNU/Linux distribution. Among many other technological legacies, Ubuntu has inherited the Debian package system. In fact, many core Ubuntu developers involved early on will credit Debian's packaging system as the reason that Debian proved such an attractive point of departure and represented its major attraction over other GNU/Linux distributions. As a result, almost all aspects of package management—from the formats to the tools—are identical on Ubuntu and Debian. In many situations, unmodified Debian packages can simply be installed on Ubuntu. In nearly all situations, unmodified Debian source packages can be built on Ubuntu. As a result, our first step is to examine an Ubuntu DEB in some depth to understand the anatomy of the package and the way it implements the features described in the preceding sections.

Source Packages

DEB source packages are clearly expressed in what is usually a three- or two-file format but may also include source packages that consist of many more files as well. This means that the package itself contains multiple files and downloading "a source package" may in fact involve downloading a small assortment of different files. Source packages can be broadly classified as either *native* DEB packages or *nonnative* DEB packages. A native DEB is a piece of software where there is no difference between the upstream version and the DEB package. In most cases, native packages are specific to either Ubuntu, Debian, or another Debian-based distribution. In other words, a native package requires no changes in order to create the package. A DEB source package will always consist of a "pristine" source archive in the form

of a gzip-compressed GNU tar file and a DSC file that will list the contents of the package and can be considered the "core" of a source package. An example DSC for a program called most that I maintain looks like this:

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

Format: 1.0
Source: most
Binary: most
Architecture: any
Version: 5.0.0a-1
Maintainer: Benjamin Mako Hill <mako@debian.org>
Standards-Version: 3.7.3
Build-Depends: debhelper (>= 4), libslang2-dev
Files:
  30f2131b67f61716f6fe1f65205da48b 155233 most_5.0.0a.orig.tar.gz
  07e3eb05ad5524fe6d885f5cdc2eb902 20160 most_5.0.0a-1.diff.gz

-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.6 (GNU/Linux)

iD8DBQFH4IoAic1LIWB1WeYRAjyOAKCrLCfuZA7b8JcvYTFYeuHrF7r34wCfVTBS
/jGUfIrELNq173sM9CorZA4=
=/Cia
-----END PGP SIGNATURE-----
```

The file is signed with a GnuPG (Gnu Privacy Guard [GPG]) or PGP (pretty good privacy) key to ensure the integrity of the file and the identity of the author. If you were to check this signature with GPG, you would see that it was signed by my GPG key. The DSC file also contains information on the version of the source format (in this case, it's the "old" format, 1.0), the name of the source package, the version of the package (split into the version of the upstream source and the version of the package after the final -), the name and e-mail of the maintainer, the architecture on which the software will run, the version of policy (marked as "standards") against which the software was created, the software that must be present to build the package, and a list of the other files this source package contains, identified by file size and MD5 hashes.

In a native DEB, there would be only a single compressed tar (tar.gz) file. In this nonnative package, there would be additional files that represent all changes to the package. This is so all the changes that the DEB packager

made are clearly visible. This is sometimes done for license reasons but is usually done just so that users can see exactly what the packager has done and what has been changed. This also makes it easy for the package maintainer to understand where a problem lies if there is an error. Changes made to a package are usually expressed in a gzip-compressed diff file that expresses all the differences between the package source and the pristine source. In the case on the previous page, it is listed as most 5.0.0a-1.diff.gz. In newer versions of the DEB source package format, additional tar files containing additions or changes to the pristine source archive are also permitted, as long as they are listed in the DSC file in the list of files.

When unpacked and with all necessary patches applied, every DEB source package will unpack into a single directory of the form packagename-version with a mandatory debian directory as a subdirectory. In the vast majority of packages, almost all changes to the source are made inside this directory. This directory contains a number of files—more than I have space to cover here. Most important among these are the control file and the rules file. The control file includes most of the information about the source package found in the DSC file (which is autogenerated using control file data) and additional information describing each binary package. The control file expresses all interpackage relationships, including Depends, Conflicts, Provides, Replaces, Recommends, Suggests, and Enhances. As the "hard" requirements, the first four are most important. Suggests and Enhances are rarely used by any program. The file also includes both a single-line and a multiline description. A sample control file (the control for most) is shown here:

```
Source: most
Section: text
Priority: optional
Maintainer: Benjamin Mako Hill <mako@debian.org>
Standards-Version: 3.7.3
Build-Depends: debhelper (>=4), libslang2-dev

Package: most
Architecture: any
Depends: ${shlibs:Depends}
Description: Pager program similar to more and less
```

The long-form description was removed from the output but in fact followed the final description line and includes text that is indented by one space and where paragraphs are separated by a single " .". As mentioned

previously, the control file consists of a series of stanzas. The first stanza will always begin with `Source:` and will include information on the source package. Each following stanza will describe a single binary package. In this case, there is only one binary package, which, like the source package, is named `most`. This situation—a single source package creating a single binary package of the same name—is a very common case.

The rules file is a GNU Make makefile and contains all of the makefile rules to create and build a package. Running debian/rules binary from within the unpacked source package directory results in the creation of a Debian package in one directory above (../) if your system has all the necessary dependencies installed. In most cases, the software will build and "install" into a series of subdirectories in the debian directory; these files in their temporary location will then be included as the package contents.

Additional files in the debian directory include the copyright file, the changelog for the package, optional scripts to be run after and before installation or removal of the package, and extra configuration data plus anything else the packager would like to include.

Binary Packages

Debian binary packages are very simple in format, so it is unnecessary to spend much time on them here. More important, they are almost never manipulated by hand. Binary packages are merely installed and removed. Changes to a binary package are made first in the source package and then new, changed binary packages are rebuilt. In Ubuntu and Debian, binary packages are a single file in an archive in the ar format. In the archive is debian-binary, which contains a series of lines, separated by newlines. At the moment, only the format version number is included. The second member of the archive is named control.tar.gz, and it contains the package control information (as described previously). The third and last member is called data.tar.gz, and it contains the file system archive as a gzipped tar archive.

Package Management in Ubuntu

The administrator of every Ubuntu installation—servers and desktops—must learn the basic mechanics of package management. As administrators need to find new software to solve particular problems, metadata in

the packaging system can be a great place to start. When administrators want to install new software, the packaging system provides the best way to do so. The Ubuntu package system will also allow users to install and remove software, check for updates—and for security updates in particular—and install these updates. Finally, when a new release of Ubuntu is made, the packaging system will allow administrators to update their systems.

Ubuntu provides a variety of different tools for package management. On a desktop Ubuntu system, users' interaction with the package management system is primarily through a little icon on the desktop that alerts them to new releases of software and through the graphical Add/Remove Programs application and a second graphical package management program called Synaptic that provides functionality to let users browse the package archives. Since these programs are covered in depth in *The Official Ubuntu Book* (ISBN: 0133017605) and because the focus of this book is servers, this section focuses on the command-line tools for package browsing and management.

Most server administrators primarily use tools in the APT family that handle high-level package management. The original tool developed for this purpose was `apt-get`. Aptitude is a frequently used alternative to `apt-get` that provides both an interactive front end and that takes most of the default `apt-get` commands. Many of the commands described in the rest of this chapter that call `aptitude` can also be used with `apt-get` with little or no difference in either output or behavior. The primary differences are in the ways that the systems resolve complicated dependency situations and certainly would not affect the reasonably simple operations described here.

Staying Up-to-Date

Each Ubuntu system stores a list of package repositories in /etc/apt/sources.list. This describes the list of "places" where your package managers—originally just APT but now several other tools—will look for updated versions of software. These sources may include local repositories on your file system, a CD in your computer, or—as is common in the vast majority of situations—a network location. To update the system's list of packages, you can run `apt-get update` or `aptitude update`.

This command downloads the latest updated package lists for all repositories listed in your /etc/apt/sources.list files and checks any cryptographic

signatures on these updates against the keys stored on your machine. On a new system, it checks only the Ubuntu package repositories that include the repositories you installed from and the security repositories.

Installing any new version of packages is as simple as running `aptitude safe-upgrade`, which is a replacement for the `apt-get upgrade` command that may be more familiar to more seasoned users. `safe-upgrade` simply tries to upgrade all installed packages to their most recent versions. Installed packages will not be removed unless they are unused, although additional packages may also be installed in order to resolve added dependencies.

APT can be configured to automatically download and upgrade packages with new versions. This is an attractive proposition to administrators who like the idea of not having to log in to their systems to keep them up-to-date. However, automatic package upgrades are subject to errors because of the particular status of software on the system or even particular configuration changes that have been made, so these automatic package upgrades can leave systems in unstable or unworkable states. As a result, automatic upgrades are neither covered in this book nor recommended by the authors.

Searching and Browsing

Historically, the primary way of searching for new packages was using the program `dselect`. Users of Ubuntu on the desktop will primarily use the Add/Remove Programs application and the graphical program Synaptic. Users on the console have several other options.

First among these is the simple program `apt-cache`, which can provide statistics about and information on packages. If, for example, I decide I want a pager like less, I can search for one in the following way:

```
$ apt-cache search pager less
less - Pager program similar to more
wdiff - Compares two files word by word
console-log - Puts a logfile pager on virtual consoles
gdesklets-data - Applets for gdesklets
jless - A file pager program, similar to more(1) supporting ISO2022
most - Pager program similar to more and less
nagios-plugins-basic - Plugins for the nagios network monitoring
  and management system
```

As you can see from the previous list, the `apt-cache search` command returned eight "hits" for my search on the two keywords *pager* and *less* and returned a list of package names followed by short one-line descriptions. The keyword search looked through the full list of available packages and focused on the package names, short descriptions, and full descriptions that are not shown in the returned list. If I want to know more about a package, `apt-cache` can also show me more about the package with the `show` subcommand as in the following example:

```
$ apt-cache show most
Package: most
Priority: optional
Section: universe/text
Installed-Size: 172
Maintainer: Ubuntu MOTU Developers <ubuntu-motu@lists.ubuntu.com>
Original-Maintainer: Benjamin Mako Hill <mako@debian.org>
Architecture: i386
Version: 5.0.0a-1
Depends: libc6 (>= 2.7), libslang2 (>= 2.0.7-1)
Filename: pool/universe/m/most/most_5.0.0a-1_i386.deb
Size: 48092
MD5sum: e089c00005b536e1b8848b7087df2bae
SHA1: 4f4ab395f340be4804732452aa112007916f90cb
SHA256:
  ccf50fb49270e7ddf7735da23e699afcd11dcfc8e241973bb17ad03bf49e6f4a
Description: Pager program similar to more and less
 Most is a paging program that displays, one windowful at a time, the
 contents of a file on a terminal. A status line at the bottom of the
 screen displays the file name, the current line number, and the
 percentage of the file so far displayed.
 .
 Unlike other paging programs, most is capable of displaying an
 arbitrary number of windows as long as they all fit on the screen,
 and different windows could be used to view the same file in
 different positions.
 .
 In addition to displaying ordinary text files, most can also display
 binary files as well as files with arbitrary ascii characters.
Bugs: mailto:ubuntu-users@lists.ubuntu.com
Origin: Ubuntu
```

You may recognize that quite a bit of this information looks like the source package information and the corresponding stanza referring to this binary

package in the control file described previously. Sure enough, this is exactly where this metadata has been extracted.

Of course, the bulk of the output is made up of the long-form description that was omitted in the previous example. There are some other fields of potential interest, including the "Original-Maintainer" or the person who packaged the system in Debian, the "Maintainer" or the person or group to contact with questions about or issues with the package, and sizes and hashes (e.g., MD5Sum, SHA1, and SHA256), which describe ways to identify that a particular version of the package was downloaded correctly and has not been modified.

Called with no arguments, Aptitude also can provide users with a Curses-based text-based interface that allows for more interactive browsing of all the packages available. For users familiar with Synaptic, this can be thought of as a text-based version of the Synaptic interface. In this mode, many search results can be navigated through with the arrow keys and different applications can be "marked" for installation.

Before concluding this tour of the options for searching and browsing for packages, it is worth pointing to the Web site at http://packages.ubuntu.com. This interface lets users search in ways that are similar to some of the tools I have shown here but with several additional useful options. In particular, the Web site lets users search for particular files in *any* package in Ubuntu. Normally, users are able to find out only which package "owns" a file if they have the package on their system. If, for example, you need a particular header file or shared library and you know only the filename, you can search on the Web site for that filename throughout all packages available in the Ubuntu archive.

Installation and Removal

Installing and removing packages is another simple task that you will do frequently. To install a package, you can invoke `apt-get` or Aptitude in a similar way, although, unlike searching, a user must be running with root privileges to do so. The recommended way to do this would be to use the `sudo` command. Since prefixing each command in this section with `sudo`

would be tedious, I have assumed the user is root, although having the user logged in as root would not be considered the best form. If I want to install most, I can simply run the following command as root:

```
# aptitude install most
Reading package lists... Done
Building dependency tree
Reading state information... Done
Reading extended state information
Initializing package states... Done
Writing extended state information... Done
The following NEW packages will be installed:
  libslang2{a} most
0 packages upgraded, 2 newly installed, 0 to remove and 0 not
  upgraded.
Need to get 0B/509kB of archives. After unpacking 1323kB will be
  used.
Do you want to continue? [Y/n/?] y
Writing extended state information... Done
Selecting previously deselected package libslang2.
(Reading database ... 362131 files and directories currently
  installed.)
Unpacking libslang2 (from .../libslang2_2.1.3-3ubuntu1_i386.deb) ...
Setting up libslang2 (2.1.3-3ubuntu1) ...
Selecting previously deselected package most.
(Reading database ... 362143 files and directories currently
  installed.)
Unpacking most (from .../most_5.0.0a-1_i386.deb) ...
Processing triggers for man-db ...
Setting up most (5.0.0a-1) ...

Reading package lists... Done
Building dependency tree
Reading state information... Done
Reading extended state information
Initializing package states... Done
Writing extended state information... Done
```

You can see in the output of the preceding command that `libslang2` was installed alongside most. In this case, Aptitude saw that most required the S-Lang library but that it was not installed. Aptitude prompted me for confirmation about the installation of the additional package (which I approved), downloaded both packages, and then installed and configured them on my system.

Removing a package is similarly simple. If I decide to remove most, I can do so by running

```
# aptitude remove most
```

In this case, `libslang2` will *not* be removed (since I have not asked for it to be removed). If I were instead to try to remove `libslang2`, Aptitude would prompt me and explain that removing `libslang2` would also require removing all of the packages that depend on it—on this system, that would just be most, but for other packages or on other systems there could be quite a few packages. This type of dependency management means that, for example, users should not (and cannot easily) remove core or essential packages. Extra "unused" packages can be removed using the command `apt-get autoremove`.

Finally, while these examples both used Aptitude, the installation and removal of packages can also be done with the lower-level tool `dpkg`. In fact, in both cases Aptitude is simply calling `dpkg` on the downloaded package files behind the scenes. Aptitude—or `apt-get`—will always download packages and work out dependencies before turning to `dpkg`. If you have already installed existing dependencies, you can install a DEB directly with `dpkg` by using the `-i` command and passing the package filename as an argument. For example, if I were given a DEB file for most, I could install it with a command like this:

```
$ dpkg -i most_5.0.0a-1_i386.deb
```

`dpkg` will check dependencies and produce an error if there are missing dependencies but will not automatically download or install packages since it does not contain the functionality to do this. I could uninstall most with `dpkg` with the command `dpkg -r most`.

Manipulating Installed Packages

`dpkg` provides dozens of methods of querying, searching, and manipulating installed packages. It contains a database of information about packages

installed on the system. To get a quick overview of what this might look like, you could run the following command:

```
$ dpkg -l most
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Cfg-files/Unpacked/Failed-cfg/Half-inst/
  trig-aWait/Trig-pend
|/ Err?=(none)/Hold/Reinst-required/X=both-problems
  (Status,Err: uppercase=bad)
||/ Name         Version      Description
+++-============-============-=============+======================
ii  most         5.0.0a-1     Pager program similar to more and less
```

Run without any arguments, `dpkg -l` will show this basic information on the installation status, name, version, and description of *every* package on your system.

Another simple task is to get a list of files contained within the package. If you have a DEB file that you have not installed, you can get this information by running `dpkg --contents` as in the following example:

```
$ dpkg --contents /var/cache/apt/archives/most_5.0.0a-1_i386.deb
drwxr-xr-x root/root        0 2008-05-06 12:06 ./
drwxr-xr-x root/root        0 2008-05-06 12:06 ./usr/
drwxr-xr-x root/root        0 2008-05-06 12:06 ./usr/bin/
-rwxr-xr-x root/root    59940 2008-05-06 12:06 ./usr/bin/most
drwxr-xr-x root/root        0 2008-05-06 12:06 ./usr/share/
drwxr-xr-x root/root        0 2008-05-06 12:06 ./usr/share/man/
drwxr-xr-x root/root        0 2008-05-06 12:06 ./usr/share/man/man1/
-rw-r--r-- root/root     5912 2008-05-06 12:06 ./usr/share/man/
  man1/most.1.gz
drwxr-xr-x root/root        0 2008-05-06 12:06 ./usr/share/doc/
drwxr-xr-x root/root        0 2008-05-06 12:06 ./usr/share/doc/most/
-rw-r--r-- root/root     2989 2007-09-09 12:14 ./usr/share/doc/
  most/changelog.gz
-rw-r--r-- root/root     5544 2008-05-06 12:06 ./usr/share/doc/
  most/copyright
-rw-r--r-- root/root     3335 2007-09-06 10:15 ./usr/share/doc/
  most/README
-rw-r--r-- root/root     1386 2006-05-01 13:51 ./usr/share/doc/
  most/lesskeys.rc
-rw-r--r-- root/root      492 2006-05-01 13:51 ./usr/share/doc/
  most/most-fun.txt
-rw-r--r-- root/root     3086 2006-05-01 13:51 ./usr/share/doc/
  most/most.rc
```

```
-rw-r--r-- root/root      2028 2008-05-06 12:06 ./usr/share/doc/most/
  changelog.Debian.gz
drwxr-xr-x root/root         0 2008-05-06 12:06 ./usr/lib/
drwxr-xr-x root/root         0 2008-05-06 12:06 ./usr/lib/mime/
drwxr-xr-x root/root         0 2008-05-06 12:06 ./usr/lib/mime/
  packages/
-rw-r--r-- root/root        94 2008-05-06 12:06 ./usr/lib/mime/
  packages/most
```

Similar information for installed packages can be retrieved with `dpkg -L`. Working in the other direction, if you have a particular file and you want to know which package "owns" it, you can use `dpkg -S` to query the database for this information. For example:

```
dpkg -S /usr/bin/most
most: /usr/bin/most
```

The binary file /usr/bin/most belongs to—no surprise here for anyone who's gotten this far—the binary package called most. Since this command is searching through each of the file lists of every package on your system, it may take some time to complete.

Manipulating Repositories

The best way to install new software in the "Ubuntu way" is never to simply download new DEB packages and install them "by hand" with `dpkg`. But APT is only kept up-to-date with the packages that it already knows about. While `dpkg` works on packages, APT works on repositories of packages that contain information on different packages, their versions, and their dependencies. As a result, to manage a package through APT, one needs to add to the system not the package, but rather the repository that contains it. This is done by adding or editing the list of "sources." While the Ubuntu desktop distribution includes a graphical tool for manipulating repositories, it can be done easily by hand, which will be the default on most systems.

The sources.list file, already mentioned several times in this chapter, is located at /etc/apt/sources.list on every Ubuntu and Debian system and is made up of a series of lines like this:

```
deb http://us.archive.ubuntu.com/ubuntu/ lucid main universe
deb-src http://us.archive.ubuntu.com/ubuntu/ lucid main universe
```

The first word will be a # symbol marking the line as a comment or else either deb or deb-src. This specifies whether the repository is a source package repository or a binary package repository. The second item is the location in the form of a URI. The third item is the name of the distribution or, as it might more accurately be described, the distribution version. In the previous example, this distribution version is lucid, which refers to the Ubuntu release of the Lucid Lynx. The remaining arguments are the lists of the components. The components provided in the core Ubuntu repositories are detailed in the following section.

An example will help illustrate the process of adding a repository. If I want to install a version of Bazaar that is always the latest released version, I will need to do this from outside the default Ubuntu repositories, which will only be updated based on the Ubuntu release cycle. Luckily, the Bazaar developers provide their own "Personal Package Repository"—a subject I'll come back to at the end of this chapter. On their Web site, they provide the deb and deb-src lines that I can simply drop into my sources.list:

```
deb http://ppa.launchpad.net/bzr/ubuntu lucid main
deb-src http://ppa.launchpad.net/bzr/ubuntu lucid main
```

If I update, I am first greeted by an error that claims that I do not have the correct cryptographic key to verify that the packages in the repository are really coming from the Bazaar developers:

```
W: GPG error: http://ppa.launchpad.net lucid Release: The following
signatures couldn't be verified because the public key is not
available: NO_PUBKEY FE8956A73C5EE1C9
```

I can easily install that by downloading the key from a trusted source like the PPA providers' Web site and saving it into a file (called /tmp/keyfile in the following example), verifying that is correct, and adding to the package manager's key database with a command such as

```
apt-key add - < /tmp/key
OK
```

The `apt-key` manual page gives more details on how keys for repositories can be managed with this useful command.

Ubuntu Default Repositories

The vast majority of packages that you will need have been packaged for Ubuntu. This is because, leveraging the work of Debian, Ubuntu provides access to a large majority of the most popular pieces of free software as packages in their own repositories.

These tens of thousands of packages are separated into a series of different sections or components. You can toggle these on and off by including them in the list of components in your sources.list. Because these have important consequences for the level of support you will receive for your software, it is worth understanding these different components so that you can decide from which areas you want to pull software. Available components on the Ubuntu server include main, restricted, universe, and multiverse. The following descriptions are adapted from the component descriptions on the Ubuntu Web site:

- **Main**
  The main distribution component contains applications that are free software, can freely be redistributed, and are fully supported by the Ubuntu team. These include the most popular and most reliable open source applications available, much of which is installed by default when you install Ubuntu. Software in main includes a hand-selected list of applications that the Ubuntu developers, community, and users feel are important and that the Ubuntu security and distribution teams are willing to support. When you install software from the main component, you are assured that the software will come with security updates and technical support.

- **Restricted**
  The restricted component is reserved for software that is very commonly used and that is supported by the Ubuntu team even though it is not available under a completely free license. Please note that it may not be possible for Ubuntu to provide complete support for this

software since the Ubuntu team is unable to fix the software but can only forward problem reports to the actual authors.

- **Universe**
  In universe one can find almost every piece of open source software and software available under a variety of less-open licenses, all built automatically from a variety of public sources. All of this software is compiled against the libraries and using the tools that form part of main, so it should install and work well with the software in main, but *it comes with no guarantee of security fixes and support*.

- **Multiverse**
  The multiverse component contains software that is not free, which means the licensing requirements of this software do not meet the "main" component license policy. The onus is on you to verify your rights to use this software and comply with the licensing terms of the copyright holder. This software is not supported and usually cannot be fixed or updated. Use it at your own risk.

Using Other Repositories

As you saw when I added the Bazaar repository several sections ago, users will still sometimes want to make use of a variety of outside repositories beyond what is provided in Ubuntu. For example, users might want to install new versions of particular applications or libraries from the development release of Ubuntu but might not want to upgrade all of their packages to the latest version.

The quasi-official "backports" repository in Ubuntu is a useful resource. It contains versions of software from the development version of Ubuntu that have been backported to install cleanly on stable versions of Ubuntu. You can add the backports by installing a DEB package by hand in a one-by-one with `dpkg` or by adding an extra line to your sources.list. Information on doing both can be found on the Ubuntu Web site at https://help.ubuntu.com/community/UbuntuBackports.

One reason that many users choose to go the à la carte method—that is, the method of downloading packages by hand and installing them with `dpkg`—as opposed to just adding the repository is because of a limitation in the

way that APT works: *APT and other tools will always install the newest version of any package available by default*. This means that if you add the backports repository, or the development repository for that matter, to your sources.list, the latest version of *everything* in that repository will be installed when you try to run an upgrade. For small repositories (like the Bazaar PPA described several sections ago that contained only Bazaar and several closely linked packages) this does not present a problem. However, in situations where you want to add a large repository of many packages like the backports repository or the development release of Ubuntu but only want a few packages, the effects will often not be what you want.

The general solution to this problem is called *pinning* or *apt pinning*. Pinning is extraordinarily powerful but, in its advanced forms, can also be very complicated. As a result, a full discussion is outside the scope of this chapter. That said, an example is shown next for the situation where I have Karmic installed but want APT to prefer packages in Lucid. To change this, I would need to create a file in /etc/apt/preferences.d that included something like the following code block:

```
Package: *
Pin: release a=karmic
Pin-Priority: 700

Package: *
Pin: release a=lucid
Pin-Priority: 600
```

Each stanza describes one release and, as is represented by the wildcard in the first line, applies to all packages. In the final line of each stanza, the pin-priority describes both relative position (i.e., in the preceding example, Karmic is preferred to Lucid) and weight that will be given to each. Weights can be tweaked so that packages will be installed, or not, except in special circumstances. Much more information on pinning is available in the `apt_preferences` manual page and in several excellent pieces of documentation on the Ubuntu and Debian wikis.

Upgrading a Whole System

A final basic task that every system administrator will need to do is to upgrade a full system. On desktop Ubuntu systems, the default way of

handling an upgrade is by using the update manager software. However, this software is designed specifically to upgrade graphical systems. Since the process can just as easily be done from the command line, that will probably be more appropriate on most servers.

In the past, upgrading most systems was a two-step process. First, the administrator would update the list of repositories (detailed in the previous section) so that references to the old release were replaced with the new release. For example, if I were upgrading from the Hardy Heron to the Gutsy Gibbon, I would replace every instance of *hardy* with *gutsy* in my source.list file. After doing this, I would run `aptitude update` exactly as I described earlier in the section "Staying Up-to-Date." This would refresh my local package metadata cache with a list of all the packages in the new distribution.

Finally, I would run `aptitude full-upgrade` which, unlike `safe-upgrade`, described previously, would upgrade all installed packages to their most recent version and would remove or install additional packages as necessary. `full-upgrade` is less conservative than `safe-upgrade` and is much more likely to perform unwanted actions. However, it is capable of upgrading packages that `safe-upgrade` cannot. Because these sorts of situations are much more common between releases, using `full-upgrade` became the recommended course for upgrading between releases. However, neither method is supported anymore.

In current releases of Ubuntu, the correct way to upgrade systems is with the `do-release-upgrade` program. `do-release-upgrade` is a script that automates the process described earlier in addition to handling a number of corner cases and exceptions intelligently. It is the supported way to upgrade one's Ubuntu server.

Mirroring a System

One common task many system administrators want to accomplish is to mirror the installed software from one machine to another. Because all software on a default Ubuntu system is installed in packages, the packaging system can make this easy. Using `dpkg`, one can get a list of all packages on the machine with the following command:

```
# dpkg --get-selections > package_list
```

This command outputs a simple list of packages and then redirects that output into a file called package_list. I can copy this file to another machine and then use it to set the list of installable packages with the following command:

```
# dpkg --set-selections < package_list
```

Finally, I can install those selections onto the target system using the following command:

```
# apt-get dselect-upgrade
```

`dselect-upgrade` is a reference to APT's predecessor `dselect` but will simply work to upgrade packages on the system and install any new packages "marked" for upgrade by `dpkg --set-selections` in the process.

Making Your Own Packages

The power of a package management system is that you can track dependencies and conflicts, do automatic upgrades, and keep track of every file on the system and which piece of software it belongs to. Installing through packages is much easier than if one simply downloads and builds from scratch, but the package management system truly shines when it comes time to uninstall or upgrade. If you've installed from source, files may be in any number of places on your file system. If you've installed from a package, removing your package will be as simple as `apt-get remove`.

As a result, many responsible system administrators find it very convenient to ensure that *all* software on their systems is installed from packages. That sounds great, but sometimes a piece of software you want—or a version of a piece of software that you want—isn't packaged or isn't built for the version of Ubuntu that you are running. The result is that you'll need to build, in one way or another, your own packages. The rest of this chapter gives a brief overview of this process and provides a starting spot for the system administrator who wants to move beyond simply consuming packages and become a producer.

Rebuilding Packages

As I hinted earlier in this chapter, many users want to rebuild existing packages as part of backporting a version of a piece of software available in

one version of Ubuntu—or Debian—to a current one. Sometimes, if an ABI has changed, a piece of software won't work on a version of Ubuntu simply because it was compiled against a set of libraries that are no longer present. This is the easiest possible case to fix because adjusting for it is simply a matter of downloading the source and rebuilding it against the new version of the libraries. This section covers doing exactly that.

Doing so will first require a source package. The source package, as you may remember from earlier in this chapter, consists of a DSC file and at least one other file. These can be downloaded as normal files from http://packages.ubuntu.com and unpacked with `dpkg-source -x` *`filename`*`.dsc`, or they can be installed automatically by using the `apt-get source` *`package`* command.

If one wanted to download and compile a package from a particular distribution—as is often the case—one could specify this explicitly with the `-t` option, which, behind the scenes, sets the default PIN for the distribution at a very high priority (990 in fact) by running (for example)

```
$ apt-get -t jaunty source --compile most
```

This would download and unpack the version of most source packages from Jaunty—assuming, of course, that the necessary `deb-src` line was included in /etc/apt/sources.list. The unpacked source code will be in a subdirectory of the current directory made up of the package name and version. In this case, the directory would be called most-5.0.0a since 5.0.0.a is the version of most that I've downloaded. When a `--compile` flag is added to the preceding `apt-get` invocation, the binary packages will also be built automatically—even if the program is in an interpreted language and there is no actual compiling taking place. If one does not use the compile flag, it can be invoked afterward in several ways. One of the simplest is by changing into the directory and then running `dpkg-buildpackage` like this:

```
$ cd most-5.0.0a
$ dpkg-buildpackage -us -uc -rfakeroot
```

This command will create an unsigned package (the `-us` and the `-uc` refer to unsigned source and unsigned changelog files) without needing root privileges (fakeroot is a program that allows packages to be built without

root). Of course, the package may also require build dependencies that are not installed by running a command in the following form:

```
# apt-get -t jaunty build-dep most
```

The `build-dep` subcommand to `apt-get` automates the process of installing all software necessary to build a given package. Running it is a frequent first step in rebuilding any package for the first time when that package is from an installed repository.

When the software in question is successfully rebuilt, the directory will contain a set of binary packages for this source package that end with .deb in the directory where it is run. In this case, the single binary package created was called most 5.0.0a-1 i386.deb. The -1 following the version number of the software refers to the version of the package and could be incremented each time we made a new version of the package. The i386 in this case simply refers to the architecture for which the binary package was built. In this case, I built it on an Intel machine. For many users, this will say amd64, which is an increasingly popular architecture. For most interpreted programs that will run on any architectures, this will say all.

New Upstream Versions

New upstream versions of packages are slightly more complicated than simply rebuilding an existing package with no modifications. Installing the package `devscripts` provides the user with a program called `uupdate` which helps with this process. To use `uupdate`, a user must first download the source package with a command like `apt-get source most`. Leave off the compile option for the moment, and then download the new upstream version tarball. There is no reason to unpack it at this point and, optionally, rename it into name-version.tar.gz format. Changing into the directory of the *old* package's source and running `uupdate` with the new upstream tarball as the argument will usually do the trick:

```
$ cd most-5.0.0a $ uupdate ../most-5.0.1.tar.gz
```

Usually, `uupdate` then deduces the version number from the upstream tarball and applies all the changes made to the old version to the new upstream

source. If `uupdate` can't decode the version number, the new version number can be specified as a second argument to the command.

The output from `uupdate` should explain the process that it follows and will end with a description of the location of the new modified source. In this case, changing to ../most-5.0.1 will put me in the new "updated" package directory. It's a good idea to look around first to make sure that things worked well. Especially it is worth checking the debian/ subdirectory and paying attention to both the control file and the changelog file in that directory, the latter of which will have been updated automatically but will probably need a little bit of tweaking. The stanza at the top will include information on the new release and can be updated or tweaked to reflect changes that you made to the file. Once you are satisfied, you can build the package with `dpkg-buildpackage` in the way described in the previous section.

Building Packages from Scratch

Building packages from scratch is much more complicated and involves getting to know quite a bit about the internals of Debian packages. As a result, it is outside the scope of this chapter. As a hint, new packages can be most easily created using the package `dh-make`, which installs the program `dh_make`, which is invoked from inside the unpacked source tarball from the upstream developer. For many simple packages, `dh_make` does most of the hard work of creating workable packages.

Much more information on creating packages for Ubuntu can be found in the Ubuntu packaging guide, which goes in depth into the process of creating packages from scratch: https://wiki.ubuntu.com/PackagingGuide.

It is worth noting one important caveat to the Ubuntu documentation: The packaging guide is focused on creating packages that are designed to be uploaded to Ubuntu. If you are creating packages that will be installed only on your own machine, the potential for harm is much less, and many of the guidelines in the packaging guide can be treated as just that—especially in the first version of a package. The difference is between workable packages and policy-compliant packages.

If you are going to proceed and create packages to be shared with others or perhaps even uploaded into the Ubuntu repositories eventually, it is a *very* good idea to follow the instructions in the packaging guidelines carefully and to use programs like lintian, which check your packages for many common errors—useful steps in any situation. If you just want things to work, a brief trip through the guide and use of `dh_make` will probably put you in good enough shape to get by.

Hosting Your Own Packages

A final step in the creation of your packages will be hosting them in a place where others can get them in the simple "add a line to your source.list file" sort of manner to which I have referred throughout this chapter. There are several different ways to do this. The easiest one and the one most commonly practiced in the Ubuntu world is to use Launchpad—the infrastructure built by Canonical and used extensively in Ubuntu's own development—to host what's called a Personal Package Archives (PPA).

With a PPA, a developer can simply upload a source package to Launchpad and the package will then be built on a variety of architectures and posted into a PPA. PPAs work exactly the same way that developing for Ubuntu does, so using them is a great preview of what you will experience if you decide to eventually upload your software in Ubuntu and get involved on the development side of things. Earlier, when I showed how to add Bazaar packages to the list of packages, I entered the list of the Bazaar PPAs. More information on PPAs is available at the following URLs: https://help.launchpad.net/Packaging/PPA and https://launchpad.net/ubuntu/+ppas.

Alternatively, you can host your own repository on your own server with any of several different tools. Although the classic tool for running these is a package called `apt-ftparchive`, the newer project `reprepro` is probably a better fit. Installing the package with that name and looking in the documentation is a good way to get started.

Index

[ ] (square brackets)
  in grep search keywords, 485–486
  in regular expressions, 466
* (asterisk)
  in shell globs, 208–209, 465
  in traceroute output, 422
  wildcard character, 465
\ (backslash)
  line continuation character, 93
  in sudoer aliases, 206
% (percent sign)
  group name indicator, 204
  Kickstart section indicator, 106
/ (slash), in IRC commands, 452
? (question mark), wildcard character, 464
@ (at sign), Kickstart task indicator, 106
# (hash mark), comment indicator
  sources.list file, 72
  Upstart, 31
^ (caret), Ctrl key symbol, 24
< (left angle bracket), redirection operator, 470–472
| (vertical line), pipe symbol, 466–467
. (dot)
  alias for current directory, 20
  package paragraph separator, 62–63
  partition separator, 94, 96
.. (dot dot), alias for directory above current, 20
> (right angle bracket), redirection operator, 470–472
>> (right angle brackets), redirection operator, 470–472
450 command, 155

A

A time, 12
a2dissite script, 139, 142
a2enmod command, 142
a2ensite script, 138, 142
aa-complain program, 209
aa-enforce program, 209
abort command, 149
Accessibility options, 4
access-key variable, 332
Account options, Kickstart, 109
Active/active clusters, 379
Active-backup or 1 mode, 373
Active-backup policy, 373
Active/passive clusters, 379
Adaptive load balancing, 374
Adaptive transmit load balancing, 374
—add command, 354
add-apt-repository command, 330
Administrator. *See* System administrator.
Advanced Package Tools (APT). *See* APT (Advanced Package Tools).
Alert escalations, 287–288
Alerts for software upgrades, 64
Aliases
  for directories, 20
  e-mail users, 147, 150
  nesting, 21
  newaliases command, 150
  sudo command, 205–206
  updating, 150
alloptions.cfg file, 85
allow command, 217
Amazon EC2. *See also* Juju.
  AMIs, picking, 320–324
  API tools, 316–319
  availability zones, listing, 319
  available AWS regions, listing, 318
  AWS Management Console, 315–316
  command-line interface, 316–319
  description, 315
  EBS (Elastic Block Storage), 321

Amazon EC2, *continued*
firewall rules, 324–325
My Account page, 315–316
post-install scripts, 329–330
prebuilt hard drive images, 320–324
presetting arguments, 318
private keys, displaying, 320
rates, 315, 319, 321
regions *vs.* availability zones, 319
registering an account, 315–316
security groups, 324–325
ssh key pairs, creating, 319–320
userdata scripts, 328–330
X.509 certificates, 316–317
Amazon EC2, instances
adding to groups, 325
creating, 321–324. *See also* Juju.
external IP, determining, 326–327
getting information about, 323–324
instance IDs, 323–324
ssh into, 326–327
starting, 327–328
stopping, 327–328
t1.micro, 315
terminating, 327–328
AMIs (Amazon Machine Images). *See also* Amazon EC2.
Canonical Ubuntu defaults, 327
definition, 321
dynamic IP addresses, 327
ebs, 321
finding, 321–322
hostname, default, 327
instance store, 321
instance types, 321
instances, creating, 321–324
user, default, 327
Apache. *See also* Web servers.
a2dissite script, 139
a2ensite script, 138
apache2ctl program, 139–141
apache2-doc package, 141
CGI scripts directory, 43, 139
configtest command, 140
configuration files, 136–137
configuration files, checking, 140–141
configuring for WordPress, 142–143
diagnostic commands, 140–141
document root directory, 43, 139
documentation, 141
environment variables for scripts, 136
/etc/apache2, 136
/etc/apache2/apache2.conf, 136
/etc/apache2/conf.d/, 137
/etc/apache2/envvars, 136
/etc/apache2/mods-available/, 137
/etc/apache2/mods-enabled/, 137–138
/etc/apache2/ports.conf, 136
/etc/apache2/sites-available/, 138
/etc/apache2/sites-enabled/, 138–139
file conventions, 136–139
fullstatus command, 140–141
graceful command, 140
graceful-stop command, 140
log files, 139
modules available to Apache, 137
multiple sites on same server, 138
port settings, 136
restart command, 139–140
restarting, 139–140
start command, 139–140
starting/stopping, 139–140
status command, 140–141
stop command, 139–140
symlinks to .load and .conf files, 137–138
symlinks to virtual hosts, 138
/usr/lib/cgi-bin/, 139
/var/log/apache2/, 139
/var/www/, 139
virtual hosts, 138
apache2 package
installing, 116
in the LAMP server package, 14
apache2ctl program, 139–141
apache2-doc package, 141
apache2-mpm-prefork package, 14
AppArmor
aa-complain program, 209
aa-enforce program, 209
complain mode, 209–210
configuration files directory, 210
enforce mode, 209–210
/etc/apparmor/, 210
/etc/apparmor.d/, 210

/etc/init.d/apparmor, 210
file conventions, 210
globs, 208–209
init script directory, 210
log directories, 210
overview, 206–207
principle of least privilege, 206–207
profiles, 207–209
rules directory, 210
/var/log/apparmor/, 210
/var/log/syslog, 210
APT (Advanced Package Tools)
apt-cache program, 65–67
apt-ftparchive package, 81
apt-get program, 58–59, 64
Aptitude program, 64
downloading packages automatically, 65
installing new package versions, 65
overview, 58–59
upgrading packages automatically, 65
apt pinning repositories, 75
apt-cache program, 65–67
apt-ftparchive package, 81
apt-get program, 58–59, 64
apt-get-upgrade command, 65
apt-install command, 103
Aptitude program
alternative to apt-get, 64
full-upgrade command, 76
installing packages, 67–68
removing packages, 69
safe-upgrade command, 65, 76
upgrade command, 65
—arch option, 304
Archiving backups, 242
Arguments, editing boot defaults, 87–88
Arguments, listing
commands, 22
init scripts, 35
installation, 5
Asterisk (*)
in shell globs, 208–209, 465
in traceroute output, 422
wildcard character, 465
at command, 478–479. *See also* cron command.
At sign (@), Kickstart task indicator, 106
Audible alarms, servers, 487
authkeys file
definition, 381
description, 386–388
node authentication, 386–388
syslog file example, 387–388
Autobuilders, 55–56
Autobuilding packages, 55–56
Auto-expiration of sudo access, 202
auto_failback option, 384
autojoin option, 382–383
Automatic
disk failure notification, 353
failback, 384
GRUB boot loader updates, 25
nodes joining clusters, 382–383
package building, 55–56
package downloading, 65
package upgrading, 65
software upgrades, 57
source discovery, Kickstart, 110
system upgrades, 76
updates, enabling, 97
Automating
chroot process, 103
Ubuntu Server installation. *See* Installing Ubuntu Server, automating; Kickstart; Preseeding.
VMs (KVM) creation, 308–309
autopsy package, 235–236
Autopsy tool, 235–236
autostart command, 310
Availability zones, 319
AWS Management Console, 315–316
AWS regions, listing, 318

B

Backing up data. *See also* BackupPC; Rescue and recovery; Restoring from backups; Snapshots.
archiving backups, 242
blackout periods, 262–263
checksum-seed option, 258
dd command, 242–244
drive imaging, 242–244
excluding directories, 259–260
frequency, 241–242
full backup interval, 261–262

Backing up data, *continued*
full backups, 241
FullAgeMax option, 262
FullKeepCnt option, 262
FullKeepCntMin option, 262
FullPeriod option, 261–262
incremental backups, 241
limiting to one file system, 258–259
pg_dump tool, 248
principles of, 240–241
with RAID, 241
retention options, specifying, 262
scheduling backups, 241, 261–263
to a separate system, 240
testing backups, 241
Backing up data, databases
MySQL, 244–248
mysqldump program, 244–248
number of backup files, specifying, 247–248
password requirements, 245, 247
pg_dump tool, 248
PostgreSQL, 248–249
scheduling, 246–249
to the screen, 244–245
Backport repositories, 74–75
Backporting, 77–79
Backslash (\)
line continuation character, 93
in sudoer aliases, 206
Backup files, location, 266
BackupPC. *See also* Backing up data; Restoring from backups.
first backup, starting, 256–257
overview, 249–250
password protection, 249–250
restore file browser, 263–264
storage requirements, 250–251
BackupPC, client machine
adding to BackupPC, 255–256
command-line interface, 256
configuring, 254–255
Web interface, 255–256
BackupPC, configuration
changing, 252–253
client machine, 254–255
command-line based, 253–254
config.pl file, 251–252
default, 251–252
SSH keys, 254
sudo, 255
Web-based, 252–253
BackupPC, rsync tweaks
backup retention, specifying, 262
blackout periods, 262–263
checksum-seed option, 258
excluding directories, 259–260
full backup interval, 261–262
FullAgeMax option, 262
FullKeepCnt option, 262
FullKeepCntMin option, 262
FullPeriod option, 261–262
host-specific tweaks, 260–261
limiting to one file system, 258–259
scheduling backups, 261–263
balance-alb or 6 mode, 374
balance-rr or 0 mode, 373
balance-tlb or 5 mode, 374
balance-xor or 2 mode, 373
Bash commands, too many arguments, 488–489
Bash history, 489
—bcast option, 308
bcast option, 383
/bin directory, 40
Binaries directories, 40–41
Binary packages
autobuilding, 55–56
creating, 55–56
installing, 56–57
overview, 63
BIND (Berkeley Internet Name Domain). *See* DNS servers, BIND.
Bind 9 DNS server, 13
BIND configuration files, documentation, 134
bind9 package, 13
bind9-doc package, 13
Black hole, redirecting files to, 478
Blackout periods, 262–263
Blk_read: total blocks read, 412
Blk_read/s: blocks read per second, 412
Blk_wrtn: total blocks written, 412
Blk_wrtn/s: blocks written per second, 412
Blogging software. *See* WordPress.
Bond modes, Ethernet bonding, 373–374
Boot arguments, editing, 87–88

Boot cheat codes, 119–121
/boot directory, 7, 42
Boot flag, setting, 13
Boot loader, partitioning, 7. *See also* GRUB.
Boot parameters, listing, 5
Boot process, GRUB boot loader
  automating updates to, 25
  changing temporarily, 25–26
  configuration file, 25
  definition, 25–26
  documentation for, 25
  internal comments, 25
  kernel options, defining, 25–26
  menu.lst file, 25
  update-grub program, 25
Boot process, kernel
  init script, 27
  initial RAM disk file, 26
  initramfs file, 26–27
  initrd file, 26
  modular kernels, 26–27
  root file system, mounting, 27
Boot process, /sbin/init program (System V init model). *See also* Upstart.
  description, 27–28
  drawbacks, 30–31
  /etc/init.d script, 28–29
  /etc/rc0.d — /etc/rc06.d scripts, 29
  /etc/rc.local script, 29
  /etc/rcS.d script, 29
  force-reload command, 29
  init scripts, 28–29
  reload command, 29
  reloading settings, 29
  restart command, 29
  runlevels, 28
  start command, 29
  starting/stopping, 29
  start-up scripts, 28–30
  status command, 29
  stop command, 29
  system states. *See* Runlevels.
  user scripts, 29
Boot process, /sbin/init program (Upstart). *See also* System V init model.
  # (hash mark), comment indicator, 31
  advantages of, 30–31
  checking job status, 32–33
  comments, 31
  default runlevel, changing, 33–34
  description, 30–31
  event-driven actions, 30–31
  script location, 31
  script syntax, 31
  start command, 32
  starting/stopping jobs, 32
  status command, 32
  stop command, 32
Boot process services, definition, 34
Boot process services, managing with init scripts
  arguments, listing, 35
  chkconfig tools, 36
  configuration, checking, 36
  configuration files, 35–36
  enabling/disabling services, 36
  extended options, 34–35
  force-reload command, 35
  PID, tracking, 36
  reload command, 35
  reloading configuration files, 35
  restart command, 35
  restarting scripts, 35
  service command, 36
  service status, checking, 35
  skeleton script, 37
  status command, 35
  symlinks, creating, 38
  update-rc.d program, 36
  writing your own, 36–38
Boot process services, managing with xinetd
  description, 38–39
  echo feature, 38
  enabling services, 39
  FTP feature, 39
  system time, displaying, 38
  TFTPD (Trivial File Transfer Protocol Daemon), 39
Boot prompts, responding to, 117
Boot screen, 3–5
"Bootable flag" field, 13
Booting LTSP
  boot option default, setting, 193
  boot option timeout, setting, 193
  boot: prompt, displaying, 193

Booting LTSP, *continued*
from the local hard drive, 193–194
from the network, 189–190
Bootloader options, Kickstart, 109
Bootstrap, Juju, 333
Botnets, 214
Bouncing e-mail messages, 153–154
Braille terminal, enabling, 4
Bridged networking, 300–302
Broadcast address, specifying, 308
broadcast or 3 mode, 373
Broadcast policy, 373
Browsing for packages, 65–67
Brute-force attacks, 213–214
Bug reporting, 459–461
BusyBox shell, 16
Bypassing installation CDs at boot, 4

C

Caches, flushing, 166
Canonical, paid support, 450–451
Caret (^), Ctrl key symbol, 24
cd command, 20
CD ejection, disabling, 100
CDs for installation. *See* Installation CDs.
CGI scripts directory, Apache Web server, 139
Chaining commands, mdadm tool, 353
Character classes, 466
Charms, Juju, 334
Cheat codes, 119–121
check command, 149
Checking job status, Upstart, 32–33
Checksums, creating, 489
checksum-seed option, 258
chgrp command, 21
chkconfig tools, 36
chkrootkit program, 236
chmod command, 21, 474
Choose a different root file system, menu option, 438
choose_interface option, 89–91
chown command, 21
chroot process, automating, 103
clean option, 431
Client connection, verifying, 416–417
Client machine, BackupPC, 254–256
Client problems *vs.* server, 416–417
Client root filesystem, 188
Client settings, defaults, 159
Clock (LTSP), setting, 187
Closed ports *vs.* firewalls, 423–424
Cloud computing. *See* Amazon EC2; Juju.
Clusters. *See also* Fault tolerance.
active/active, 379
active/passive, 379
adding hosts to, 279
defining, 276–277
fencing, 380
forcibly killing a server, 380
host status, determining, 379
monitoring nodes. *See* Heartbeat tool.
overview, 378
quorum, 379
replicated storage. *See* DRBD.
resource descriptions, 384–385
separate connection for node monitoring, 380
shooting the other node in the head, 380
split-brain syndrome, 379
Command-line administration
becoming root, 24
editing files, 23–24
nano editor, 23–24
sudo command, 24
vi editor, 23–24
Command-line administration, directories
. (dot), alias for current, 20
.. (dot dot), alias for directory above current, 20
aliases, 20–21
cd command, 20
changing, 20
current, 19–20
current, listing files in, 18–19
group, displaying, 20
home, confirming, 18
information about, listing, 19–20
last modification time, displaying, 20
links, displaying, 20
ls command, 18–20
moving around the system, 18–21
name, displaying, 20
ownership, displaying, 20
permissions, displaying, 20

pwd command, 18
size, displaying, 20
symlinks, identifying, 20
Command-line administration, files
chgrp command, 21
chmod command, 21
chown command, 21
groups, 20–21
information about, listing, 19–20
last access time, displaying, 20
name, displaying, 20
ownership, 21
permissions, 21
size, displaying, 20
symlinks, 20
Command-line administration, running processes
killing, 22–23
monitoring in real time, 21–23
PID, finding, 22–23
ps command, 21–23
stopping, 21–23
top command, 21
Commands. *See also* specific commands.
arguments, listing, 22
bash history, 489
pathname, finding, 486
repeating, 487
vs. services, 288
wildcard characters, 488
Commenting out configuration lines, 375, 376
Comments
(hash mark), comment indicator, 31, 72
GRUB boot loader, 25–26
Upstart, 31
Common section, DRBD configuration file, 390
Communication timeout, setting, 383
Comparing files, 489
Complain mode, AppArmor, 209–210
config.pl file, 251–252
configtest command, 140
configuration. *See specific programs*.
configuration files. *See specific programs*.
configurator tool, Kickstart, 111
Contact list, configuring, 289–290
Contacts, configuring, 287
Control file, source packages, 62–63
—copy option, 308
Copying
packages to another system, 77
SSH key files, 308
Copying files
from non-RAID disks to RAID, 356
from RAID 1 to RAID 5, 362
CPU, monitoring
idle time, 407
load, 272
system time, 407
user time, 407
create command, 166
—create option, 350
createdb command, 170
createuser command, 170
Critical thresholds, setting, 286
cron command, 480–483. *See also* at command.
crontabs, 483
Cryptographic keys, repositories, 72–73
cupsys package, 14
cupsys-bsd package, 14
Current directory
. (dot), alias for, 20
identifying, 18
listing files in, 19–20
Current load, analyzing, 313–315

D

-d option, 304
Database servers. *See* MySQL databases; PostgreSQL databases.
Databases, backing up
MySQL, 244–248
mysqldump program, 244–248
number of backup files, specifying, 247–248
password requirements, 245, 247
pg_dump tool, 248
pg_dumpall tool, 248
PostgreSQL, 248–249
psql tool, 248
scheduling, 246–249
to the screen, 244–245
Databases, Tripwire
default directory, 232
"file does not exist" message, 229

Databases, Tripwire, *continued*
initializing, 229–230
"unknown file system type" message, 229
updating, 230–231
Databases, used by Samba, 176
dd command, 242–244, 445, 486
ddrescue command, 445
deadtime option, 383
DEB (Debian) format. *See* Package management, DEB format.
debconf database, dumping, 85
debconf-get-selections, 85
default command, 216–217
DEFAULT keyword, 193
Default runlevel, changing, 33–34
Defense in depth, 201
defoma package, 14
Degraded arrays, 355
deity. *See* APT (Advanced Package Tools).
delete allow command, 217
delete command, 220
delete deny command, 217
Deleted files, recovering, 440–443
Deleting. *See also* Removing.
hosts, 289
mail queue messages, 149
services, 289
user accounts, PostgreSQL, 170
deny command, 217
denyhosts program, 213–214
Dependency checking, package management, 57, 59–60
Desktop alerts for software upgrades, 64
Destination directory, specifying, 304
Destination files, list of, 308
destroy command, 310
—detail argument, 352
—detail —scan command, 350–351
/dev directory, 44
device command, 110
Device files, 477–478
Device files directory, 44
Device information directory, 45
/dev/mem, 478
/dev/null, 478
/dev/random, 478
devscripts package, 79
/dev/ttyS0, 478
/dev/urandom, 478
df command, 413–416
DHCP (Dynamic Host configuration Protocol)
automating Ubuntu Server installation, 118–123
leases, list of, 161
selection by static leases, 121–122
selection by subnet, 123
timeout duration, setting, 90
timing out, 90
DHCP servers
coexisting with LTSP, 191–192
configuration files, 161
DHCP leases, list of, 161
dynamic configuration, 161–162
/etc/dhcp3/dhcpd.conf, 161
file conventions, 161
installing, 160
log files, 161
overview, 160
setting up for PXE boot server, 112–113
static configuration, 162–163
/var/lib/dhcp3/dhcpd.leases, 161
/var/log/syslog, 161
dh-make program, 80–81
Diagnostic commands, 140–141
diff command, 489
dig command, 50
dig tool, 420, 492
dir command (Windows). *See* ls command.
Direct restore, 264
Directories. *See also* File system hierarchy; *specific directories*.
aliases, 20–21
cd command, 20
changing, 20
command-line administration. *See* Command-line administration, directories.
excluding from backups, 259–260
group, displaying, 20
information about, listing, 19–20
last access time, 12
last modification time, 20
links, displaying, 20

ls command, 18–20
moving around the system, 18–21, 489–490
moving back to previous, 489–490
name, displaying, 20
noatime option, 12
ownership, displaying, 20
permissions, displaying, 20
pwd command, 18
Samba, sharing, 176–177
size, displaying, 20
symlinks, 20
A time, 12
variable size, directory for, 43
Directories, current
. (dot), alias for, 20
identifying, 18
listing files in, 19–20
disable command, 216
Disabled users. *See* Accessibility options.
Disk partitioning. *See also* Installing Ubuntu Server.
administrator options, 7–13
/boot directory, 7
for the boot loader, 7
for dual-boot capability, 8
grouping partitions or disks, 8
Guided, LVM, 8
Guided, with entire disk, 8
/home directory, 6
Kickstart, 110
with Kickstart, 105–108
KVM VMs, 306
for LTSP, 187
MD (multidisk) devices, 350
migrating from RAID 1 to RAID 5, 359–362, 364–365
migrating non-RAID disks to RAID, 355, 358
/opt directory, 6
partitions, definition, 5–6
partitions, maximum per disk, 10
for personal files for user accounts, 6
for RAID devices, 346–349
resizing current partitions, 8
for temporary files, 7
for third-party programs, 6
/tmp directory, 7
/usr directory, 7
/var directory, 6
for variable-size data, 6
Disk partitioning, manual
allocating free space, 9–10
boot flag, setting, 13
"Bootable flag" field, 13
extended partitions, 10
file system, specifying, 10
file system settings, 10–13
initializing a blank drive, 9
inodes, setting number of, 12
inside extended partitions, 10
"Label" field, 12
logical partitions, 10
mount options, 11–12
"Mount options" field, 11–12
mount point, specifying, 11
"Mount point" field, 11
naming partitions, 12
partition size, specifying, 10
partitions as physical volumes, 10
primary partitions, 10
"Reserved blocks" field, 12
reserving space for the superuser, 12
"Typical usage" field, 12
"Use as" field, 10–11
Disk partitioning, preseeding
custom schemes, 92–94
expert_recipe for, 92–94
formatting partitions, 94
LVM partitions, 95–96
maximal size, 93
minimal size, 93
mountpoint, specifying, 94
overview, 91
partman-auto/choose_recipe option, 92
partman-auto/method option, 91
partman-auto/purge_lvm_from_device option, 91–92
partman/choose_partition option, 92
partman/confirm option, 92
partman-lvm/confirm option, 91–92
partman-partitioning/confirm_write_new_label option, 92

Disk partitioning, preseeding, *continued*
primary partition, 94
priority, specifying, 93–94
warning prompts, disabling, 91
Disk space
allocating, 9–10
freeing, rescue and recovery, 430
reserving for the superuser, 12
Disk space, troubleshooting
df command, 413–415
du command, 414–415
excessive tmp files, 415
full file system, 415
out of inodes, 415–416
usage, by directory, 414–415
usage, by file system, 414–415
Diskless clients, 181, 183
Disks. *See also* Hard drives.
erasing, 486
failure, automatic notification, 353. *See also* Hard drives, rescue and recovery.
images, restoring from, 243
I/O, monitoring, 272
management, drbdadm command, 396–397
snapshots of, 8
DISPLAY option, 193
Distributions
definition, 53
specifying, 78
Dividing the problem space, 400–401
DNS (Domain Name System)
address, specifying, 308
administration, documentation, 134
status, checking, 420–421
ufw firewall example, 221
—dns option, 308
DNS queries, dig tool, 492–493
DNS servers
caching name server, 129
definition, 13
DNS master, 129–132
DNS slave, 132–133
host e-mail address, specifying, 130
overview, 126–127
SOA (Start of Authority), specifying, 130
TTL (Time To Live), default setting, 130
DNS servers, BIND
configuration files, 128
configuration files, reloading, 134
current status, checking, 134
default log file, 129
/etc/bind/, 128
/etc/bind/db.*, 128
/etc/bind/named.conf, 128
/etc/init.d/bind9, 128
file conventions, 127–129
flush command, 134
init script, location, 128
installing, 127
managing with rndc, 134
as name server, 127
named.conf file, 128
reconfig command, 134
reload command, 134
retransfer zone command, 134
server caches, flushing, 134
slave zone files, location, 128
status command, 134
/var/cache/bind, 128
/var/log/syslog, 129
working directory, 128
zones, reloading, 134
DNS servers, BIND zone files
adding, 129–132
location, 128
ownership, 131
permissions, 131
referencing in named.conf, 131–132
reloading, 134
retransferring, 134
DNS slave server
configuring the master server, 132–133
configuring the slave server, 133
overview, 132
Document root directory, 139
Documentation. *See also* Help and resources.
Apache Web server, 141
BIND configuration files, 134
DNS administration, 134
DNS servers, BIND GRUB boot loader, 25
doc files, 458
expert_recipe partitioning, 92–94

GroundWork front end, 296
installation CDs, 4
localhost, 457–458
man command, 22
man pages, 457–458
mdadm tool, 354
The Official Ubuntu Book, 64
online, 456
packages, 54–55
sudoers file, 204
troubleshooting problems and solutions, 402–403
Domain default, specifying, 307
Domain name for sent mail, 152
Domain Name System (DNS). *See* DNS (Domain Name System).
—domain option, 307
Domains, accepting mail from, 152
do-release-upgrade program, 76
Dot (.)
alias for current directory, 20
package paragraph separator, 62–63
partition separator, 94, 96
Dot dot (..), alias for directory above current, 20
Dovecot, 157–158
Downloading packages automatically, 65
dpkg option, 431
dpkg program
copying packages to another system, 77
file owner package, identifying, 71
listing installed packages, 76–77
listing package files, 70–71
manipulating installed packages, 69–71
mirroring a system, 76–77
overview, 55, 69
querying installed packages, 69–71
searching installed packages, 69–71
DRBD
configuring Heartbeat, 395–396
drbddisk script, 395–396
initializing resources, 393–395
installing, 389–393
for NFS, 396
overview, 388–389
for Samba, 396
DRBD, drbdadm command
disk management, 396–397
drbd.conf file, changing, 397
initializing resources, 393–395
replacing failed disks, 397
solving split-brain problem, 398
DRBD configuration file, creating
common section, 390
example, 389–390
global section, 390
internal metadisk, 391–392
resource section, 390–391
split-brain policy, changing, 392
drbdadm command
disk management, 396–397
drbd.conf file, changing, 397
initializing resources, 393–395
replacing failed disks, 397
solving split-brain problem, 398
drbd.conf file
changing, 397
condensed, 393
initializing, 393–395
drbddisk script, 395
Driver information directory, 45
Drives. *See* Disks; Hard drives.
drop command, 166
dropuser command, 170
DSA keys, OpenSSH servers, 160
dselect program, 58–59, 65
du command, 414–415
Dual boot
LTSP servers, 192–194
partitioning for, 8
duck command, 414–415
Duplicate lines, removing from sorted output, 469–470
Dynamic configuration, DHCP servers, 161–162
Dynamic Host configuration Protocol (DHCP). *See* DHCP (Dynamic Host Configuration Protocol).
Dynamic preseeding
chain loading files, 101–102
overview, 100–101
preseed/early_command option, 103
preseed/late_command option, 103–104

Dynamic preseeding, *continued*
preseed/run option, 102–103
running custom commands, 102–104

E

EBS (Elastic Block Storage), 321
ebs AMIs, 321
ec2-add-keypair command, 320
ec2-api-tools package, 317–319
ec2-authorize command, 324–325
ec2-describe-availability-zones command, 319
ec2-describe-group command, 324–325
ec2-describe-instances command, 323–324
ec2-describe-regions command, 318
ec2din command, 323–324, 326
ec2dre command, 318
ec2kill command, 328
ec2-revoke command, 325
ec2run command, 321–324, 328–330
ec2-run-instances command, 321–324
ec2start command, 328
ec2stop command, 328
echo feature, 38
Editing
boot arguments, 87–88
command-line administration, 23–24
nano editor, 23–24
preseed.cfg file, 87–88
Tripwire policies, 227–228
vi editor, 23–24
Edubuntu. *See* LTSP (Linux Terminal Server Project).
802.3ad or 4 mode, 373
E-mail. *See also* Mail.
bounced messages, avoiding, 153–154
mail servers, 14, 144. *See also* POP/IMAP servers; Postfix mail server.
sending a test via telnet, 490–491
sending notifications, 291
storing, 156–157
E-mail, example
configuration file, 151–153
domain name for sent mail, 152
domains, accepting mail from, 152
Internet host name, 152
mailbox size limit, setting, 153
mailbox_size_limit option, 153
mydestination option, 152
myhostname option, 152
mynetworks option, 152–153
myorigin option, 152
networks, relaying mail, 152–153
open relays, 153
overview, 150
relayhost option, 152
routing outbound mail, 152
spam exposure, 153
enable command, 216
Encryption. *See also* OpenSSH servers; SSH security.
LVM (Logical Volume Manager), 372
Tripwire settings, 232
Enforce mode, AppArmor, 209–210
Environment variables directory, 225
Environment variables for scripts, 136
Erasing disks, 486
Escalations, 287–288
/etc directory, 42–43
/etc/aliases, 147
/etc/apache2, 136
/etc/apache2/apache2.conf, 136
/etc/apache2/conf.d/, 137
/etc/apache2/envvars, 136
/etc/apache2/mods-available/, 137
/etc/apache2/mods-enabled/, 137–138
/etc/apache2/ports.conf, 136
/etc/apache2/sites-available/, 138
/etc/apache2/sites-enabled/, 138–139
/etc/apparmor/, 210
/etc/apparmor.d/, 210
/etc/backuppc, 265
/etc/backuppc/apache.conf, 265
/etc/backuppc/config.pl, 251–252, 265
/etc/backuppc/hosts, 265
/etc/backuppc/htpasswd, 265
/etc/bind/, 128
/etc/bind/db.*, 128
/etc/bind/named.conf, 128
/etc/defaults/ufw, 225
/etc/dhcp3/dhcpd.conf, 161
/etc/dovecot/, 158
/etc/exports, 177

/etc/fstab file, pointing to arrays, 357, 362
/etc/hosts directory, 47
/etc/init.d script, 28–29
/etc/init.d/apparmor, 210
/etc/init.d/backuppc, 265
/etc/init.d/bind9, 128
/etc/init.d/dovecot, 158
/etc/init.d/mysql, 165
/etc/init.d/nfs-kernel-server, 178
/etc/init.d/nmdb, 175
/etc/init.d/postfix, 148
/etc/init.d/postgresql, 171
/etc/init.d/smdb, 175
/etc/init.d/ssh, 160
/etc/init.d/ufw, 225
/etc/init/nmbd.conf, 175
/etc/init/smdb.conf, 175
/etc/mysql/, 164
/etc/mysql/conf.d/, 165
/etc/mysql/debian-cnf, 165
/etc/mysql/debian-start, 165
/etc/mysql/my.cnf, 164
/etc/network/interfaces directory, 46
/etc/postfix/, 146
/etc/postfix/main.cf, 146–147
/etc/postgresql/, 170
/etc/postgresql/9.1/main/pg_hba.conf, 171
/etc/postgresql/9.1/main/pg_ident.conf, 171
/etc/postgresql/9.1/main/postgresql.conf, 171
/etc/rc0.d — /etc/rc06.d scripts, 29
/etc/rc.boot, removing, 227
/etc/rc.local script, 29
/etc/rcS.d script, 29
/etc/resolv.conf directory, 47
/etc/samba/, 174
/etc/samba/smb.conf, 175
/etc/ssh/, 159
/etc/ssh/ssh_config, 159
/etc/ssh/sshd_config, 159
/etc/ssh/ssh_host_dsa_key, 160
/etc/ssh/ssh_host_dsa_key.pub, 160
/etc/ssh/ssh_host_rsa, 160
/etc/ssh/ssh_host_rsa.pub, 160
/etc/tripwire/, 232
/etc/tripwire/*-local.key, 232
/etc/tripwire/*-site.key, 232
/etc/tripwire/tw.cfg, 232
/etc/tripwire/twcfg.txt, 232
/etc/tripwire/tw.pol, 232
/etc/tripwire/twpol.txt, 227, 232
/etc/ufw/, 224–225
/etc/ufw/after6.rules, 225
/etc/ufw/after.rules, 225
/etc/ufw/before6.rules, 225
/etc/ufw/before.rules, 225
Etherboot booting, 181
Ethernet bonding. *See also* Fault tolerance.
  802.3ad or 4 mode, 373
  active-backup or 1 mode, 373
  active-backup policy, 373
  adaptive load balancing, 374
  adaptive transmit load balancing, 374
  balance-alb or 6 mode, 374
  balance-rr or 0 mode, 373
  balance-tlb or 5 mode, 374
  balance-xor or 2 mode, 373
  bond modes, 373–374
  broadcast or 3 mode, 373
  broadcast policy, 373
  commenting out configuration lines, 375, 376
  enabling the bonded interface, 377–378
  IEEE 802.3ad Dynamic link aggregation, 373
  ifenslave package, installing, 375–377
  log entry, example, 378
  new bond device, example, 377–378
  overview, 372–373
  round-robin policy, 373
  testing fail-over, 378
  XOR policy, 373
Ethernet bonding, network configuration
  Ubuntu 10.04, 375–376
  Ubuntu 12.04 and newer, 376–377
Ethernet devices, labeling, 46
ethtool program, 417
Event-driven actions, 30–31
exec command, 488
exec option, 32
—execscript option, 308
Execute a shell in /dev/sdal, menu option, 436
Execute a shell in the installer environment, menu option, 437
Execute permission, 472–473

exit command, 170
expert_recipe for preseeding partitioning, 92–94
Extended options, init scripts, 34–35
Extended partitions, 10
extended-status command, 167

F

F1-F6, key functions, 4–5
Failed disks, replacing, 352–354, 397
Failed logins, monitoring, 213–214
Fail-over, testing, 378
fakeRAID, 344
fakeroot program, 78–79
Fault tolerance
  hard drives. *See* RAID (Redundant Array of Inexpensive Disks).
  hot-swapping components, 342
  Juju, 337–338
  LVM (Logical Volume Manager), 372
  networks. *See* Ethernet bonding.
  principles, 342–343
  quick response time, 343
  redundancy, 342–343
  single points of failure, eliminating, 343
  techniques. *See* Clusters; Ethernet bonding; RAID (Redundant Array of Inexpensive Disks).
Favoring past solutions, 401–402
Fencing, 380
File cache, monitoring, 271
File conventions. *See specific programs*.
File servers. *See also* NFS; Samba.
  overview, 174
  role of, 15
File system hierarchy, core directories. *See also* Directories; Files.
  /bin, 40
  /boot, 42
  core binaries, 40
  core system libraries, 40
  /dev, 44
  device and driver information, 45
  device files, 44
  /etc, 42–43
  generic mount location, 44
  GRUB configuration files, 42
  /home, 43–44
  home directories, 43–44
  intramfs files, 42
  kernel images, 42
  /lib, 40
  /media, 44
  /mnt, 44
  non-critical binaries and libraries, 41
  /opt, 42
  /proc, 44–45
  removable media, mount location, 44
  /root, 44
  root user, home directory, 44
  /sbin, 40
  spool files, 43
  /sys, 45
  system configuration files, 42–43
  system logs, 43
  temporary file storage, 45
  third-party programs, 42
  /tmp, 45
  user home directories, 43–44
  /usr, 41
  /usr/bin, 41
  /usr/lib, 41
  /usr/local, 41
  /usr/sbin, 41
  /var, 43
  variable size files and directories, 43
  /var/log, 43
  /var/spool, 43
  /var/www, 43
  virtual file systems, 45
  Web server's directories, 43
File systems
  formatting, MD (multidisk) devices, 350
  root, mounting, 27
  settings, 10–13
  specifying for disk partitions, 10
  unintentionally erasing, 433
  will not unmount, 490
File systems, rescue and recovery
  corrupted, 432–433
  fsck tool, 432–433
  fstab file mistakes, 433–434
  mount command, 432

mounted, displaying, 432
primary superblocks missing, 433
umount command, 432
unintentionally erasing, 433
UUID, discovering, 434
UUID changed, 433–434
won't mount, 432–434
File types
device files, 477–478
hard links, 476–477
inodes, 476
symlinks, 475
Files. *See also specific files.*
checksums, creating, 489
chgrp command, 21
chmod command, 21
chown command, 21
command-line administration. *See* Command-line administration, files.
comparing, 489
groups, 21
information about, listing, 19–20
last access time, 12, 94
last modification time, 20
MAC (Modify, Access, Change) times, 12
name, displaying, 20
noatime option, 12
open, listing, 490
ownership, 21, 472–473
permissions, 21
redirecting to a null device, 478
search and replace text, 487–488
size, displaying, 20
symlinks, 20
temporary storage, directory for, 45
A time, disabling, 12
for user accounts, partitioning for, 6
variable size, directory for, 43
variable size, partitioning for, 6
Files, copying
from non-RAID disks to RAID, 356
from RAID 1 to RAID 5, 362
Files, in packages
integrity verification, 58
listing, 70–71
owned by, identifying, 71
owner package, identifying, 71
source packages, 60–63
Files, permissions
changing, 474
chmod command, 474
execute, 472–473
groups, listing, 473
groups command, 473
read, 472–473
types of, 472–473
write, 472–473
Filtering input to pipes, 467–468
find command, 488
Firewalls. *See also* ufw program, firewalls.
vs. closed ports, 423–424
detecting, 423–424
hardware, 215
Kickstart, 110
layers of protection principle, 215
overview, 214–216
software, 215
ufw command, 424
Firewalls, rules
Amazon EC2, 324–325
denying by default, 217
hacking, 214
listing, 424–425
undoing, 217
—firstboot option, 309
—firstlogin option, 309
—flavour option, 303–304
Floating IPs, 379
fls tool, 441–443
flush command, 134, 148–149
flush-* commands, 166
Flushing
DNS server caches, 134
mail queues, 148–149
MySQL caches and settings, 166
foomatic-db package, 14
foomatic-filters package, 14
force-reload command, 29, 35
Forensic analysis, 235–236
Forensics tools, 440–443
format option, 94
Formats, packages, 52

Formatting
partitions, preseeding, 94
RAID arrays, 361
450 command, 155
Free software repositories, 73
fsck tool, 432–433
fstab file mistakes, 433–434
FTP feature, 39
Full backups, scheduling, 241, 261–262
Full file system, 415
FullAgeMax option, 262
FullKeepCnt option, 262
FullKeepCntMin option, 262
FullPeriod option, 261–262
fullstatus command, 140–141
full-upgrade command, 76
Fully-supported software repositories, 73

G

Ganglia monitor. *See also* Monitoring, tools for.
gmond program, 273–276
installing, on all hosts, 274–276
local RRD files, 274
mcast_join option, 275
overview, 273–274
port option, 275
Ganglia server
clusters, adding hosts to, 279
clusters, defining, 276–277
configuring, 276–278
gmetad program, 273–274, 276–278
grids, defining, 277–278
Ganglia Web front end
clusters, adding hosts to, 279
ganglia-webfrontend package, 278–279
installing, 278–279
monitor duration, changing, 279
ganglia-monitor package, 274–276
ganglia-webfrontend package, 278–279
Gateway access, verifying, 418–419
Gateway address, specifying, 308
Global section, DRBD configuration file, 390
Globs, AppArmor, 208–209
gmetad program, 273–274, 276–278
gmond program, 273–276
gpart tool, 443–444
graceful command, 140
graceful-stop command, 140
Graphical console, VMs (KVM), 313–314
Graphing aggregate statistics. *See* Ganglia.
grep command
[ ] (square brackets), in search keywords, 485–486
filtering input to pipes, 467–468
in search results, 485–486
Greylisting, 154–156
Grids, defining, 277–278
GroundWork. *See* Nagios, GroundWork front end.
Group-based access, sudo command, 202
Groups
chgrp command, 21
configuring, 288
displaying, 20
files, 21
hosts, 288
listing, 473
membership, displaying, 473
membership default, 98
partitions or disks, 8
groups command, 473
GRUB
automating updates to, 25
boot device, specifying, 99
changing temporarily, 25–26
configuration file, 25
configuration files directory, 42
default setup, 99
definition, 25
description, 25–26
documentation for, 25
internal comments, 25
kernel options, defining, 25–26
menu.lst file, 25
migrating from RAID 1 to RAID 5, 359
password protection, 100
reading from RAID 5 arrays, 355
rescue and recovery, 437–438
update-grub program, 25
updating, rescue and recovery, 431
GRUB, manual install
migrating from RAID 1 to RAID 5, 366
migrating non-RAID disks to RAID, 358–359

grub option, 431
GRUB2, 25
Guess Partition tool, 443–444
Guided partitioning, 8
—gw, 308

H

ha.cf file
  auto_failback option, 384
  autojoin option, 382–383
  bcast option, 383
  deadtime option, 383
  definition, 381
  example, 382–384
  initdead option, 383
  keepalive option, 383
  logfacility option, 384
  node option, 384
  ping option, 384
  respawn option, 384
  warntime option, 383
halt command, 39
Handicapped users. *See* Accessibility options.
Hard drives. *See also* Disks.
  erasing, 486
  failed, replacing, 352–354, 397. *See also* Hard drives, rescue and recovery.
  grouping, 8. *See also* Disk partitioning.
  health, monitoring, 268–269
  imaging, 242–244
  partitioning. *See* Disk partitioning.
  prebuilt images for Amazon EC2, 320–324
  setting as faulty, 353
  statistics monitoring, 272
  testing, 426–427
Hard drives, rescue and recovery
  dd command, 445
  ddrescue command, 445
  drbdadm command, 396–397
  imaging drives, 445–447
  imaging partitions, 446
  mdadm tool, 352–354
  replacing failed disks, 352–354, 397
  scanning for problems, 444–447
  storing drive images, 445–447
Hard links, 476–477
Hardware. *See also specific hardware*.
  firewalls, 215
  interrupts, 407
  KVM VMs, 313–314
  RAID, 344
  troubleshooting. *See* Troubleshooting, hardware.
Hardware/software hybrid RAID. *See* fakeRAID.
haresources file
  cluster resource descriptions, 384–385
  definition, 381
  description, 384–385
Hash mark (#), comment indicator
  sources.list file, 72
  Upstart, 31
Headless server, installing Ubuntu Server on, 5
Hearing impaired users. *See* Accessibility options.
Heartbeat tool
  automatic failback, 384
  cluster example, 381
  communication timeout, setting, 383
  configuration files, 381. *See also specific files*.
  configuring, main methods, 380
  configuring for DRBD, 395–396
  installing, 382
  ipfail script, starting, 384
  network connectivity, gauging, 384
  overview, 380
  seconds between heartbeats, setting, 383
  service loading timeout, setting, 383
  syslog facility, specifying, 384
Heartbeat tool, authkeys file
  definition, 381
  description, 386–388
  node authentication, 386–388
  syslog file example, 387–388
Heartbeat tool, ha.cf file
  auto_failback option, 384
  autojoin option, 382–383
  bcast option, 383
  deadtime option, 383
  definition, 381
  example, 382–384
  initdead option, 383
  keepalive option, 383
  logfacility option, 384
  node option, 384

Heartbeat tool, ha.cf file, *continued*
ping option, 384
respawn option, 384
warntime option, 383
Heartbeat tool, haresources file
cluster resource descriptions, 384–385
definition, 381
description, 384–385
Heartbeat tool, nodes
automatically joining clusters, 382–383
communication, 383
manual definition, 384
Help and resources. *See also* Rescue and recovery; Troubleshooting.
bug reporting, 459–461
Canonical, paid support, 450–451
general Ubuntu help, 452
installation CDs, 4–5
IRC (Internet Relay Chat), 452–455
Juju, 339
Launchpad project, 459–461
LoCo (Local Community) Teams, 458
mailing lists, 455–456
man command, 22
mouse-over for option help, 293
online support options, list of, 461
in other languages, 459
#ubuntu, 452
#ubuntu-server, 452
Web forums, 451
XChat program, 452–455
Help and resources, documentation
doc files, 458
localhost, 457–458
man pages, 457–458
online, 456
Hesios, 109
hi: hardware interrupts, 407
High I/O wait, troubleshooting, 411–413
High-contrast screen option, 4
history command, 489
Holding mail queue messages, 149–150
/home directory
confirming, 18
description, 43–44
partitioning, 6
Host definitions, BackupPC, 265
Host network address, specifying, 307
Host status, determining, 379
Host-based access, sudo command, 202
Hosting your own packages, 81
Hostname
AMIs, default, 327
LTSP, specifying, 187
for mail servers, 152
translating to IP address. *See* DNS servers.
VM, specifying, 304
—hostname option, 304
Hosts
adding, 291–292
defining, 47–48
deleting, 289
grouping, 288
MAC address, determining, 162–163
service checks, adding, 291
Hosts, Groundwork
profiles, 287
selecting, 287
settings, specifying, 287
Hot-swapping components, 342

I

icat tool, 441–443
ICMP blocked, 423
id: CPU idle time, 407
IDSs (intrusion detection systems), 226–227. *See also* Tripwire.
IEEE 802.3ad Dynamic link aggregation, 373
ifconfig command
configuring network interfaces, 49
confirming network configuration, 418
determining MAC addresses, 162–163
network card errors, 426
"Ifconfig: command not found" message, 49
ifdown command, 48
ifenslave package, installing, 375–377
ifup command, 48
Imaging
hard drives, 445–447
partitions *vs.* whole drives, 446
servers, 234
Immediate reboot, disabling, 100

Incident response
autopsy package, 235–236
Autopsy tool, 235–236
chkrootkit program, 236
forensic analysis, 235–236
imaging the server, 234
prosecuting the intruder, 233
pulling the plug, 233–234
redeploying the server, 234–235
root kits, checking for, 236
Sleuth Kit tools, 235–236
sleuthkit package, 235–236
Incremental backups, 241
Init scripts. *See also* System V init model; Upstart.
drawbacks, 30–31
kernel boot process, 27
networking, 30–31
rescue and recovery, 434
respawning, 30
restarting, 35
Init scripts, managing services
arguments, listing, 35
chkconfig tools, 36
configuration, checking, 36
configuration files, 35–36
enabling/disabling services, 36
extended options, 35
force-reload command, 35
PID, tracking, 36
reload command, 35
reloading configuration files, 35
restart command, 35
restarting scripts, 35
service command, 36
service status, checking, 35
status command, 35
symlinks, creating, 38
update-rc.d program, 36
writing your own, 36–38
initctl command, 32–33
initdead option, 383
Initial RAM disk file, 26
Initializing
blank drives, 9
DRBD resources, 393–395
Tripwire databases, 229–230
initramfs file, 26–27
initrd file, 26
Inodes
hard links, 476
running out of, 415–416
setting number of, 12
Input, redirecting, 471–472
Installation CDs
bypassing at boot, 4
checking for defects, 4
documentation, 4
getting, 2–3
help, 4–5
as rescue disks, 4
Installer console, 15–16
Installing
binary packages, 56–57
DHCP servers, 160
DNS servers, BIND, 127
Dovecot, 157–158
DRBD, 389–393
Ganglia monitor, 274–276
Ganglia Web front end, 278–279
Heartbeat, 382
Juju, 330–332
KVM. *See* KVM, installing.
mdadm tool, 349
MySQL, 163–164
new packages, 65, 67–68
OpenSSH servers, 159
Postfix, 144–145
PostgreSQL, 14, 169–170
Postgrey, 155–156
Samba, 174
ufw program, 216
WordPress, 142
Installing Ubuntu Server. *See also* Disk partitioning.
accessibility options, 4
arguments, listing, 5
Bind 9 DNS server, 13
bind9 package, 13
bind9-doc package, 13
boot parameters, listing, 5
boot screen, 3–5
Braille terminal, enabling, 4
BusyBox shell, 16

Installing Ubuntu Server, *continued*
cupsys package, 14
cupsys-bsd package, 14
defoma package, 14
F1-F6, key functions, 4–5
foomatic-db package, 14
foomatic-filters package, 14
on a headless server, 5
high-contrast screen option, 4
install mode, selecting, 4
installation log, viewing, 15–16
installation options, 4–5
installer console, 15–16
keyboard mapping, 4
keyboard modifiers, enabling, 4
language, specifying, 3–4, 4
memory, testing, 4
on-screen keyboard, 4
openssh-server package, 13
Postfix mail server, 14
postgresql package, 14
rebooting the system, 16
samba package, 15
samba-doc package, 15
screen magnifier, enabling, 4
screen reader, enabling, 4
server BIOSs, 3
smbfs package, 15
winbind package, 15
without a monitor, 5
Installing Ubuntu Server, automating. *See also* Kickstart; Preseeding; PXE boot server deployment.
boot cheat codes, 119–121
DHCP approach, benefits of, 118
DHCP selection, by subnet, 123
DHCP selection, static leases, 121–122
multiple Kickstart files, 118–119
overview, 84, 118
pxelinux menu, changing, 118
unattended install, 118
Installing Ubuntu Server, server roles
DNS, 13
LAMP, 14
mail server, 14
OpenSSH, 13
PostgreSQL database, 14
print server, 14
Samba file server, 15
instance store AMIs, 321
in-target command, 103
Internal metadisk, 391–392
Internet, as troubleshooting reference, 403
Internet host name, for mail servers, 152
Internet Relay Chat (IRC), 452–455
Internet site option, 145
Internet with smarthost option, 145
intramfs files, directory for, 42
Intrusion detection systems (IDSs), 226–227. *See also* Tripwire.
I/O wait, 407
iostat program, 411–412
iotop program, 413
IP addresses
changing, LTSP, 194–195
displaying, 50
LTSP, tied to MAC addresses, 191–192
translating hostnames to. *See* DNS servers.
—ip option, 307
ipchains program, 215
ipfail script, starting, 384
iptables, rules directory, 225
iptables program, 215
IRC (Internet Relay Chat), 452–455

J

Java, Tomcat server, 15
JeOS, 302
Job status, querying with Upstart, 32–33
Juice. *See* JeOS.
Juju
access credentials, 331–332
access key pairs, 331–332
bootstrap, 333
charms, 334
configuring, 330–332
deploying services, 333–337
destroying entire environment, 339
destroying services, 338–339
fault tolerance, 337–338
installing, 330–332
online resources, 339

overview, 330
status, displaying, 333
~/.juju/environments.yaml file, 331
juju bootstrap command, 331, 333
juju deploy command, 334–337
juju deploy mysql command, 334–337
juju destroy-environment command, 339
juju destroy-service command, 338
juju expose command, 337
juju status command, 333, 335–337
juju unexpose command, 337

K

keep option, 94
keepalive option, 383
Keeping it simple
security principle, 200
SSH security, 215–216
troubleshooting principle, 401
Kerberos 5, Kickstart support, 109
Kernel boot process
init script, 27
initial RAM disk file, 26
initramfs file, 26–27
initrd file, 26
modular kernels, 26–27
root file system, mounting, 27
Kernel flavor, specifying, 303–304
Kernel images, directory for, 42
Kernel options, defining with GRUB boot loader, 25–26
Key-based authentication, 211–213
Keyboard
layout, LTSP, 186
mapping at installation, 4
modifiers, enabling, 4
on-screen, 4
Keys, Tripwire, 227, 232
Kickstart. *See also* Installing Ubuntu Server; Preseeding; PXE boot server deployment.
@ (at sign), task indicator, 106
% (percent sign), section indicator, 106
account options, 109
automated source discovery, 110
bootloader options, 109
configurator tool, 111
configuring for a CD-ROM, 104–108
device command, 110
excluding %packages packages, 110
firewalls, 110
Hesios, 109
initial user settings, 109
Kerberos 5, 109
launching, 105
LDAP, 109
lilo options, 109
limitations, 109–110
local disk support, 110
multiple files, 118–119
new options, 108–109
NFS support, 110
overview, 104
package group names, 110
partitioning, 105–108, 110
%post section scripts, 110–111
%pre section scripts, 110–111
preseed option, 108–109
root password, disabling, 109
root privileges, enabling, 109
rootpw command, 109
running custom commands, 110–111
Samba authentication, 109
shell scripts, 110
supplemental driver disk, 110
system-config-kickstart package, installing, 105
user command, 109
xconfig —monitor option, 110
kill command, 167
Killing processes
MySQL, 167
by PID, 22–23
Postfix, 149
KVM, installing
KVM packages, 299–300
prerequisites, 298–299
setting up users, 299
support BIOS, enabling, 299
testing the installation, 299–300
ubuntu-vm-builder script, 299. *See also* vmbuilder tool.
virsh command, 299–300
virtualization extensions, confirming, 298

KVM, network configuration
—bcast option, 308
bridged networking, 300–302
broadcast address, specifying, 308
default setup, 300
defaults, configuring, 300
DNS address, specifying, 308
—dns option, 308
domain default, specifying, 307
domain option, 307
—domain option, 307
gateway address, specifying, 308
—gw, 308
host network address, specifying, 307
—ip option, 307
—mask option, 307
—net option, 307
static IP address, assigning, 307
subnet mask, specifying, 307
wireless adapters, bridging support, 302
KVM virtual machines. *See* VMs (KVM).

L

"Label" field, 12
LABEL localboot option, 193
LAMP servers, 14
Language, specifying, 3–4, 186
Last access time
directories, 12
files, 12, 94
logging, 94
Launchpad
bug reporting, 459
help and resources, 459
hosting your own packages, 81
Layers of protection principle, 201, 215
LDAP, in Kickstart, 109
Left angle bracket (<), redirection operator, 470–472
less program, 23
—level option, 350
/lib directory, 41
Lib/ufw/user6.rules, 225
/lib/ufw/user.rules, 225
—libvirt option, 304
Licensed software repositories, 74
lilo options, in Kickstart, 109
Links, displaying, 20
Linux Terminal Server Project (LTSP). *See* LTSP (Linux Terminal Server Project).
Listing
boot parameters, 5
directory information, 19–20
files in current directory, 19–20
firewall rules, 424–425
installed packages, 76–77
mail queue messages, 149
open files, 490
package files, 70–71
processes, MySQL, 166
Listing arguments in
commands, 22
init scripts, 35
installation, 5
Ln command
hard links, creating, 476
symlinks, creating, 475
Local devices for LTSP, 195–196
Local disk support, Kickstart, 110
Local keys directory, Tripwire, 232
Local only option, 146
Localhost
documentation, 457–458. *See also* Help and resources.
troubleshooting. *See* Troubleshooting, localhost.
LoCo (Local Community) Teams, 458
Log directories
AppArmor, 210
Tripwire, 233
Log entry, example, 378
Log files
Apache Web server, 139
BackupPC, 266
DHCP servers, 161
DNS servers, BIND, 129
Dovecot, 158
MySQL, 165
NFS, 178
OpenSSH servers, 160
Postfix, 147–148
PostgreSQL, 171
Samba, 176

syslog facility, 384
syslog file example, 387–388
system logs, directory for, 43
logfacility option, 384
Logging access, sudo command, 203
logging command, 217
Logging last access time, 94
Logical partitions, 10
Logical storage. *See* LVM (Logical Volume Manager).
Loopback (lo) interface, 46
ls command, 18–20
lsof command, 490
LTSP (Linux Terminal Server Project)
availability in Ubuntu, 183
benefits of, 182–183
boot process, technical details, 181–182
diskless clients, 181, 183
and Edubuntu, 180
Etherboot booting, 181
initial booting, 181
overview, 180–181
PXE booting, 181
required network cards, 181
thin clients, 183
LTSP servers, configuring
boot option default, setting, 193
boot option timeout, setting, 193
boot: prompt, displaying, 193
booting from the local hard drive, 193–194
booting from the network, 189–190
client root filesystem, 188
clock, 187
DEFAULT keyword, 193
DISPLAY option, 193
first user, creating, 188
hostname, 187
initial client setup, 189–190
initial server setup, 188–189
IP addresses, changing, 194–195
keyboard layout, 186
LABEL localboot option, 193
language, 186
for local devices, 195–196
NICs (network interface cards), 184–186
partitioning the hard disk, 187
PROMPT option, 193
screen resolution, 188
sound, 197
SOUND=True statement, 197
SSH keys, updating, 195
TIMEOUT option, 193
for USB devices, 195–196
LTSP servers, installing
coexisting with a DHCP server, 191–192
on a desktop, 190–191
for dual boot, 192–194
IP addresses, tied to MAC addresses, 191–192
prerequisites, 183–184
reservations, 191–192
in Ubuntu, 190–191
from the Ubuntu alternate CD, 186–188
LTSP servers, password suppression, 195
LVM (Logical Volume Manager)
encryption, 372
fault tolerance, 372
Guided partitioning, 8
LVs (logical volumes), 370
overview, 369
partitions, preseeding, 95–96
PEs (physical extents), 370
PVs (physical volumes), 370
redundancy, 372
setting up, 371–372
theory of, 370
VGs (volume groups), 370
LVs (logical volumes), 370

M

MAC address, determining, 162–163
MAC (Modify, Access, Change) times, 12
Magnifier (screen), enabling, 4
Mail queues. *See also* E-mail.
flushing, 148–149
postqueue command, 149
privileged operations on, 149
status, checking, 149
Mail queues, messages
deleting, 149
hold time before bouncing, 154
holding, 149–150
listing, 149

Mail servers, 14, 144. *See also* POP/IMAP servers; Postfix mail server.
Mail spool directory, 147
Mailbox size limit, setting, 153
mailbox_size_limit option, 153
Maildirs, enabling, 156–157
mail.err file, 147
mail.info file, 147
Mailing lists, 455–456
mail.log file, 147
mail.warn file, 147
Main repositories, 73
Man pages, 457–458. *See also* Help and resources.
Managing
  DNS servers, BIND, 134
  packages. *See* Package management.
  services, with xinetd, 38–39
Managing boot process services, with init scripts
  arguments, listing, 35
  chkconfig tools, 36
  configuration, checking, 36
  configuration files, 35–36
  enabling/disabling services, 36
  extended options, 34–35
  force-reload command, 35
  PID, tracking, 36
  reload command, 35
  reloading configuration files, 35
  restart command, 35
  restarting scripts, 35
  service command, 36
  service status, checking, 35
  status command, 35
  symlinks, creating, 38
  update-rc.d program, 36
  writing your own, 36–38
Managing boot process services, with xinetd
  description, 38–39
  echo feature, 38
  enabling services, 39
  FTP feature, 39
  system time, displaying, 38
  TFTPD (Trivial File Transfer Protocol Daemon), 39
Managing services with init scripts
  arguments, listing, 35
  chkconfig tools, 36
  configuration, checking, 36
  configuration files, 35–36
  enabling/disabling services, 36
  extended options, 34–35
  force-reload command, 35
  PID, tracking, 36
  reload command, 35
  reloading configuration files, 35
  restart command, 35
  restarting scripts, 35
  service command, 36
  service status, checking, 35
  status command, 35
  symlinks, creating, 38
  update-rc.d program, 36
  writing your own, 36–38
Managing VMs (KVM)
  autostart command, 310
  current load, 313–315
  destroy command, 310
  graphical console, 313–314
  hardware, 313–314
  power off, 310
  RAM, changing, 311–312
  remote management, 313–315
  restore command, 310
  resume command, 310–311
  resuming, 310–311
  rolling back to snapshots, 310
  save command, 310
  setmaxmem command, 311
  setmem command, 311
  shutdown command, 310
  shutting down, 310
  snapshotting, 310, 313
  start command, 309–310
  starting at boot time, 310
  starting the VM, 309–310
  suspend command, 310–311
  suspending current state, 310–311. *See also* Snapshots.
  virsh command, 309–312
  virt-manager utility, 312–315
Manual partitioning. *See also* Disk partitioning.
  allocating free space, 9–10
  boot flag, setting, 13

"Bootable flag" field, 13
extended partitions, 10
file system settings, 10–13
initializing a blank drive, 9
inodes, setting number of, 12
inside extended partitions, 10
"Label" field, 12
logical partitions, 10
mount options, 11–12
"Mount options" field, 11–12
mount point, specifying, 11
"Mount point" field, 11
naming partitions, 12
partition size, specifying, 10
primary partitions, 10
"Reserved blocks" field, 12
reserving space for the superuser, 12
"Typical usage" field, 12
"Use as" field, 10–11
Manuals. *See* Documentation.
—mask option, 307
Mauelshagen, Heinz, 369
mcast_join option, 275
MD (multidisk) devices, creating and using
after installation, 349–350
file system, 350, formatting
during installation, 347–348
mounting, 350
number of active devices, specifying, 350
partitions, specifying, 350
RAID level, specifying, 350
md5sum tool, 489
mdadm tool
chaining commands, 353
—create option, 350
creating MD devices, 350–351
—detail argument, 352
—detail —scan command, 350–351
disk failure, automatic notification, 353
disks, setting as faulty, 353
documentation, 354
installing, 349
—level option, 350
—query argument, 352
—raid-devices option, 350
replacing a failed disk, 352–354
resyncing swapped disks, 354
software RAID management, 351–354
swapping disks, 352–354
/media directory, 44
Memory. *See* RAM.
Memtest86+ tool, 427–428
menu.lst file, 25
Metrics, troubleshooting localhost
hi: hardware interrupts, 407
id: CPU idle time, 407
ni: nice CPU time, 407
si: software interrupts, 407
st: steal time, 407
sy: system CPU time, 407
system load average, 404–406
top command, 406–408, 410
us: user CPU time, 407
wa: I/O wait, 407
Migrating from RAID 1 to RAID 5
booting GRUB, 359
copying files to new system, 362
creating mount points, 361
destroying original, 364
disk partitioning, 359–362, 364–365
/etc/fstab file, pointing to arrays, 362
formatting RAID arrays, 361
general procedure, 360–366
GRUB, manual install, 366
overview, 359
partitions, detecting as RAID devices, 366
"partitions contain a file system" warning, 361
rebooting, 363
rescue disk, 359
starting at boot, 362
syncing arrays, 366
update-initramfs argument, changing, 362
Migrating to software RAID from non-RAID disks
adding original partitions, 358
changing the UUID, 358
copying files to new system, 356
creating arrays, 355
degraded arrays, 355
disk partitioning, 355
/etc/fstab file, pointing to arrays, 357
GRUB, manual install, 358–359
overview, 354–355

Migrating to software RAID from non-RAID disks, *continued*
partitions, detecting as RAID devices, 359
rebooting, 357–358
rescue disc, 355
starting at boot time, 356–357
syncing arrays, 358
temporary mount points, 356
mirror/country option, 97
Mirroring
preseeding, 96–98
RAID disks, 345
a system, 76–77
the Ubuntu archive, 116
/mnt directory, 44
Modify, Access, Change (MAC) times, 12
Modular kernels, 26–27
Monitor, installing Ubuntu Server without, 5
Monitoring
aggregating statistics. *See* Ganglia.
alerts. *See* Nagios.
CPU load, 272–273
disk I/O, 272–273
disk statistics, 272–273
drive health, 268–269
file cache, 271
memory, 271
multicast IP traffic, 276
network I/O, 272
performance, 272
RAM statistics, 269–273
running processes, in real time, 21–23
selected time periods, 272–273
swap cache, 271
system load, 269–273
trending. *See* Nagios.
Monitoring, tools for
alerts. *See* Nagios.
ganglia-monitor package, 274–276. *See also* Ganglia.
sar tool, 270–273
smartd daemon, 269
Smartmontools, 268–269
sysstat package, 269–273
tcpdump program, 276
trending. *See* Nagios.
mount command, file system rescue and recovery, 432
Mount location, 44
Mount options, partitions, 11–12
"Mount options" field, 11–12
"Mount point" field, 11
Mount points
migrating from RAID 1 to RAID 5, 361
migrating non-RAID disks to RAID, 356
partitions, 11
preseeded partitions, specifying, 94
temporary, 356
Mounted file systems, displaying, 432
Mounting
MD (multidisk) devices, 350
root file system, 27
mountpoint option, 94
Mouse-over for option help, 293
Moving around the system, 18–21
MTA (Mail Transport Agent), 227
Multicast IP traffic, monitoring, 276
Multidisk (MD) devices. *See* MD (multidisk) devices.
Multiverse repositories, 74
My Account page, 315–316
mydestination option, 152
myhostname option, 152
mynetworks option, 152–153
myorigin option, 152
mysql command, 246
MySQL databases. *See also* PostgreSQL databases.
configuring for WordPress, 143
create command, 166
creating/deleting, 166
current status, checking, 165
database files, location, 165
drop command, 166
/etc/init.d/mysql, 165
/etc/mysql/, 164
/etc/mysql/conf.d/, 165
/etc/mysql/debian-cnf, 165
/etc/mysql/debian-start, 165
/etc/mysql/my.cnf, 164
extended-status command, 167
file conventions, 164–165
files, 165

flush-* commands, 166
init script, 165
installing, 163–164
kill command, 167
log files, 165
mysqladmin tool, 165–167
overview, 163
password command, 166
passwords, 14, 164, 166
phpMyAdmin program, installing, 167–168
process management scripts, 165
processes, 166–167
processlist command, 167
removing, 166
restoring from backups, 246
settings, flushing, 166
status, checking, 167
status command, 167
ufw firewall example, 222
/var/lib/mysql/, 165
/var/log/syslog, 165
Web administration, 167–168
mysqladmin tool, 165–167
mysql-client-5.0 package, 14
mysqldump program, 244–248
mysql-server-5.0 package, 14

N

Nagios, GroundWork front end
administration password, changing, 284
Apache installation, 282
configuration files, 283
configuring, 283–286
core directory, 282
description, 280–281
documentation, 296
/etc/init.d/groundwork, 283
file conventions, 282–283
host status, checking, 285–286
init script, 283
initial host scan, 284–285
installing, 281–282
/usr/local/groundwork, 282
/usr/local/groundwork/apache2, 282
/usr/local/groundwork/nagios, 283
Nagios configuration
advanced, 292–295
alert escalations, 287–288
commands *vs.* services, 288
committing changes, 288, 289
contact list, 289–290
contacts, 287
control, 288
escalations, 287–288
groups, 288
mouse-over for option help, 293
notifications, enabling, 290–291
notifications, sending via e-mail, 291
overview, 286–289
performance analysis, 289
services, deleting, 289
services *vs.* commands, 288
settings, exporting, 289
time periods, 288
tools, 289
Nagios configuration, hosts
adding, 291–292
deleting, 289
grouping, 288
profiles, 287
selecting, 287
service checks, adding, 291
settings, specifying, 287
Nagios configuration, service checks
adding, 286, 291
creating, 294–295
critical thresholds, setting, 286
default settings, overriding, 293
options, changing, 286
warning thresholds, setting, 286
Name servers
defining, 47
DNS servers as, 127
inaccessible, 420–421
not configured, 420–421
problems, troubleshooting, 420–421
named.conf file, 128, 131–132
Names of files and directories, displaying, 20
Naming partitions, 12
Native DEB packages, 60–63
—net option, 307

Netboot tarball, 113–115
netstat command, 424
Network card errors, troubleshooting, 425–426
Network configuration
  Ubuntu 10.04, 375–376
  Ubuntu 12.04 and newer, 376–377
Network interface cards (NICs), 181, 184–186
network option, 431
Networking
  connectivity, gauging, 384
  core programs, 48–50
  dig command, 50
  Ethernet devices, labeling, 46
  ifconfig command, 49
  ifdown command, 48
  ifup command, 48
  information about, getting, 48–49
  interface configuration, verifying, 418
  I/O, monitoring, 272
  IP address, displaying, 50
  under KVM. *See* KVM, network configuration.
  loopback (lo) interface, 46
  nslookup command, 50
  open relays, 153
  relaying mail, 152–153
  route command, 49
  settings, checking and changing, 48–49
  status, checking, 48–49
  troubleshooting. *See* Troubleshooting networks.
Networking, configuration files
  for all networking devices, 46–47
  /etc/hosts, 47
  /etc/network/interfaces, 46
  /etc/resolv.conf, 47
  hosts, defining, 47–48
  name servers, defining, 47
New bond device, example, 377–378
newaliases command, 150
NFS. *See also* Samba.
  configuration files, 177
  configuration sample, 178–179
  DRBD, 396
  /etc/exports, 177
  /etc/init.d/nfs-kernel-server, 178
  file conventions, 177–178
  init script, 178
  Kickstart discovery, 110
  log files, 178
  overview, 177
  root squashing, disabling, 179
  ufw firewall example, 223–224
  user permissions, 179
  /var/log/syslog, 178
  "Wrong file system type" message, 179
ni: nice CPU time, 407
NICs (network interface cards), 181, 184–186
nmap command, 423–424
No configuration option, 145
noatime option, 12, 94
Node authentication, 386–388
node option, 384
Nodes, cluster
  automatically joining clusters, 382–383
  communication, 383
  manual definition, 384
Nonnative DEB packages, 60–63
NOPASSWD: statement, 205
Notifications
  enabling, 290–291
  sending via e-mail, 291
nslookup command, 50, 420–421

O

The Official Ubuntu Book, 64
Online documentation, 456. *See also* Help and resources.
On-screen keyboard, 4
OOM (out-of-memory) killer, 410–411
Open relays, 153
Open source software repositories, 74
OpenSSH servers. *See also* SSH security.
  client settings, defaults, 159
  configuration files, 159
  definition, 13
  DSA keys, 160
  /etc/init.d/ssh, 160
  /etc/ssh/, 159
  /etc/ssh/ssh_config, 159
  /etc/ssh/sshd_config, 159
  /etc/ssh/ssh_host_dsa_key, 160
  /etc/ssh/ssh_host_dsa_key.pub, 160
  /etc/ssh/ssh_host_rsa, 160

/etc/ssh/ssh_host_rsa.pub, 160
file conventions, 159–160
init script, 160
installing, 159
log files, 160
overview, 158–159
RSA keys, 160
server settings, defaults, 159
/var/log/auth.log, 160
openssh-server package, 13
/opt directory
description, 42
partitioning, 6
Original packager, displaying, 67
Out-of-memory issues troubleshooting, 409–411
Out-of-memory (OOM) killer, 410–411
Output, redirecting, 470–472
Ownership
chown command, 21
directory, displaying, 20
files, 472–473
files, displaying, 20
zone files, 131

P

Package management. *See also* APT (Advanced Package Tools); Repositories.
Aptitude program, 64, 67–69
autobuilders, 55–56
automatic software upgrades, 57
basic functions, 55–58
binary packages, 55–57
browsing for packages, 65–67
DEB format. *See* Package management, DEB format source packages.
dependency checking, 57, 59–60
desktop alerts, 64
do-release-upgrade program, 76
dselect program, 58–59, 65
file integrity verification, 58
formats, 52
front end programs, 58–59
full-system upgrades, 75–76
for in-development software, 60
installing new versions, 65, 67–68
mirroring a system, 76–77
original packager, displaying, 67
package information, getting, 65–67
package maintainer, identifying, 67
package statistics, getting, 65–67
repositories, listing, 64–65
RPM format, 52
searching for packages, 65–67
shared library upgrades, 57
show subcommand, 66–67
smart upgrades, 59
staying current, 64–65
Synaptic, 64–65
tools for, 64–67, 69. *See also* specific tools.
uninstalling packages, 57, 59, 69
VMs (KVM). *See* VMs (KVM), package management.
Package management, DEB format
binary packages, 55–56, 63
introduction, 52
overview, 60
Package management, DEB format source packages
autobuilding, 55–56
control file, 62–63
definition, 55
files contained in, 62–63
native DEB packages, 60–63
nonnative DEB packages, 60–63
rules file, 62–63
unpacking, 62
Package management, dpkg program
copying packages to another system, 77
file owner package, identifying, 71
listing installed packages, 76–77
listing package files, 70–71
manipulating installed packages, 69–71
mirroring a system, 76–77
overview, 69
querying installed packages, 69–71
searching installed packages, 69–71
Packages. *See also specific packages.*
autobuilders, 55–56
background, 53
browsing for, 65–67
building automatically, 55–56
contents, 54
copying to another system, 77

Packages, *continued*
description, 53–55
distributions, 53
documentation, 54–55
downloading automatically, 65
files, 70–71
group names, Kickstart, 110
information, getting, 65–67
installed, 69–71
maintainer, identifying, 67
metadata, 54
original packager, displaying, 67
%packages, excluding, 110
preseeding, 96–98
rebuilding, 77–79
removing, 69
rescue and recovery, 431
for server roles, determining, 13
for server roles, listing, 15
statistics, getting, 65–67
uninstalling, 57, 59, 69
upgrading, 76
upgrading automatically, 65
Packages, making your own
apt-ftparchive package, 81
backporting, 77–79
devscripts package, 79
dh-make program, 80–81
fakeroot program, 78–79
guidelines, 80–81
hosting, 81
Launchpad, 81
new upstream versions, 79–80
overview, 77
PPAs (Personal Package Archives), 81
rebuilding packages, 77–79
reprepro project, 81
from scratch, 80–81
specifying a distribution, 78
Ubuntu packaging guide, 80
uupdate program, 79–80
without root permissions, 78–79
—part option, 306
Partially-supported software repositories, 73–74
Partition tables, restoring, 443–444
Partitions. *See also* Disk partitioning; Preseeding, partitions.
creating for RAID devices, 346–349
definition, 5–6
detecting as RAID devices, 359, 366
imaging for rescue and recovery, 446
maximum per disk, 10
resizing, 8
root, 6
swap space, 6
"Partitions contain a file system" warning, 361
partman-auto/choose_recipe option, 92
partman-auto/method option, 91
partman-auto/purge_lvm_from_device option, 91–92
partman/choose_partition option, 92
partman/confirm option, 92
partman/confirm_nooverwrite option, 92
partman-lvm/confirm option, 91–92
partman-partitioning/confirm_write_new_label option, 92
—pass option, 304
Passphrases, 211–213, 227
passwd command, 435
password command, 166
Passwordless access to rules, 203
Passwords
authentication, 202, 211–213
backing up databases, 245, 247
BackupPC, 249–250, 265
default, specifying, 304
disabling, 213
GroundWork administration, changing, 284
GRUB, 100
MySQL, 14, 164, 166
plain-text authentication, 158
prompt, bypassing, 213
removing, 205
resetting, 435
Samba, 175
suppressing display of, 195
Patches, security, 201
Pathnames, shortcuts to, 486
Percent sign (%)
group name indicator, 204
Kickstart section indicator, 106
Performance, monitoring, 272

Performance analysis, Nagios, 289
Period (.). *See* Dot (.).
Permissions
  chmod command, 21
  directory, displaying, 20
  files, 21
  Postfix, checking, 149
  root user, assuming, 24
  users, NFS, 179
  zone files, 131
Personal Package Archives (PPAs), 81
PEs (physical extents), 370
pg_dumpall tool, 248
php5-mysql package, 14
phpMyAdmin program, installing, 167–168
phppgadmin package, 172
phpPgAdmin tool, 172–173
Physical volumes, partitions as, 10. *See also* PVs (physical volumes).
PID
  finding, in running processes, 22–23
  killing processes by, 22–23
  tracking with init scripts, 36
ping command, 419
ping option, 384
Pinning repositories, 75
Pipes
  | (vertical line), pipe symbol, 466–467
  filtering input to, 467–468
  grep command, 467–468
  overview, 466–470
  removing duplicate lines, 469–470
  sort command, 468–469
  sorting input to, 468–469
  uniq command, 469–470
Policy files, Tripwire, 227–228, 232
POP/IMAP servers. *See also* E-mail; Mail servers.
  Dovecot, 157–158
  e-mail, storing, 156–157
  Maildirs, enabling, 156–157
  overview, 156
  ufw firewall example, 222
port option, 275
Ports
  configuring, 136
  first serial, 478
  listening, testing, 424
  remote, testing, 423–424
%post section scripts, 110–111
Postfix mail server
  450 command, 155
  abort command, 149
  administering, 148–150
  bounced messages, avoiding, 153–154
  check command, 149
  configuration files, 146–147
  configuration files, reloading, 148
  configuration types, 145–146
  current status, checking, 149
  /etc/aliases, 147
  /etc/init.d/postfix, 148
  /etc/postfix/, 146
  /etc/postfix/main.cf, 146–147
  file conventions, 146–148
  flush command, 148–149
  greylisting, 154–156
  init script, 148
  installing, 14, 144–145
  Internet site option, 145
  Internet with smarthost option, 145
  killing processes, 149
  Local only option, 146
  log files, 147–148
  mail spool directory, 147
  No configuration option, 145
  permissions, checking, 149
  Postgrey, installing and configuring, 155–156
  postsuper command, 149
  relay_domains option, 153–154
  reload command, 148
  Satellite system option, 146
  secondary servers, 153–154
  spam exposure, 154–156
  status command, 149
  user alias mappings, 147, 150
  user mailbox directory, 147
  /var/log/mail.*, 147–148
  /var/spool/mail/, 147
  /var/spool/postfix/, 147
Postfix mail server, example
  configuration file, 151–153
  domain name for sent mail, 152

Postfix mail server, example, *continued*
domains, accepting mail from, 152
Internet host name, 152
mailbox size limit, setting, 153
mailbox_size_limit option, 153
mydestination option, 152
myhostname option, 152
mynetworks option, 152–153
myorigin option, 152
networks, relaying mail, 152–153
open relays, 153
overview, 150
relayhost option, 152
routing outbound mail, 152
spam exposure, 153
Postfix mail server, mail queue messages
deleting, 149
hold time before bouncing, 154
holding, 149–150
listing, 149
Postfix mail server, mail queues
flushing, 148–149
postqueue command, 149
privileged operations on, 149
status, checking, 149
PostgreSQL databases. *See also* MySQL.
authentication information, 171
backing up, 248–249
configuration files, 170–171
createdb command, 170
createuser command, 170
databases, setting up, 170
dropuser command, 170
/etc/init.d/postgresql-9.1, 171
/etc/postgresql/, 170
/etc/postgresql/9.1/main/pg_hba.conf, 171
/etc/postgresql/9.1/main/pg_ident.conf, 171
/etc/postgresql/9.1/main/postgresql.conf, 171
exit command, 170
file conventions, 170–171
init script, 171
installing, 14, 169–170
log files, 171
overview, 168–169
phppgadmin package, 172
phpPgAdmin tool, 172–173
restoring from backups, 248
super user account, setting up, 169–170
ufw firewall example, 223
user accounts, creating/deleting, 169–170
usernames, mapping to PostgreSQL usernames, 171
/var/log/postgresql/, 171
Web-based administration, 171–173
postgresql package, 14
Postgresql package, installing, 169–170
Postgrey
default configuration, 155–156
installing and configuring, 155–156
reloading settings, 156
smtpd_recipient_restrictions option, 155
tweaking the whitelist, 155–156
Post-install scripts, 308–309, 329–330
postqueue command, 149
postsuper command, 149
Pound sign (#), comment indicator
sources.list file, 72
Upstart, 31
Power on/off, 310
PPAs (Personal Package Archives), 81
%pre section scripts, 110–111
preseed option, 108–109
preseed.cfg file
creating, 85–89
default, setting up, 86–87
editing, 87–88
example, 86
preseed/early_command option, 103
preseed_fetch command option, 103–104
Preseeding. *See also* Installing Ubuntu Server; Kickstart; PXE boot server deployment.
automatic updates, enabling, 97
CD ejection, disabling, 100
choose_interface option, 89–91
configuring for CD-ROM, 85–89
custom package repositories, 97–98
debconf database, dumping, 85
debconf-get-selections, 85
default boot arguments, editing, 87–88
default user account, disabling, 99

group membership default, 98
immediate reboot, disabling, 100
mirrors, 96–98
networking options, 89–91
options, displaying, 85
options, shorthand for, 88
overview, 84–85
packages, 96–98
preserving server data, 86–87
root account, enabling, 98
UID default, 98
user settings, 98–99
Preseeding, dynamic
apt-install command, 103
chain loading files, 101–102
chroot process, automating, 103
installing extra packages, 103–104
in-target command, 103
overview, 100–101
preseed/early_command, 103
preseed_fetch command option, 103–104
preseed/late_command, 103–104
preseed/run option, 102–103
running custom commands, 102–104
running custom programs, 103
setting dynamic values, 103
Preseeding, GRUB
boot device, specifying, 99
default setup, 99
password protection, 100
Preseeding, partitions
custom schemes, 92–94
expert_recipe for, 92–94
formatting partitions, 94
LVM partitions, 95–96
maximal size, 93
minimal size, 93
mountpoint, specifying, 94
overview, 91
partman-auto/choose_recipe option, 92
partman-auto/method option, 91
partman-auto/purge_lvm_from_device option, 91–92
partman/choose_partition option, 92
partman/confirm option, 92
partman/confirm_nooverwrite option, 92
partman/confirm_write_new_label option, 92
partman-lvm/confirm option, 91–92
primary partition, 94
priority, specifying, 93–94
warning prompts, disabling, 91
Preseeding, preseed.cfg file
creating, 85–89
editing, 87–88
example, 86
preseed/late_command option, 103–104
preseed/run option, 102–103
Preserving server data, 86–87
Primary partitions, 10, 94
Principle of least privilege, 200–201, 206–207
Print servers, installing, 14
Priority of preseeded partitions, specifying, 93–94
Privileged operations on mail queues, 149
Privileges. *See* Permissions.
/proc, editing, 228
/proc directory, 44–45
Processes
command-line administration. *See* Command-line administration, running processes.
listing, MySQL, 167
monitoring in real time, 21–23
PID, finding, 22–23
ps command, 21–23
stopping, 21–23
top command, 21
Processes, killing
MySQL, 167
by PID, 22–23
Postfix, 149
processlist command, 167
Processor architecture, specifying, 304
/proc/mdstat file, 351–354
Profiles
AppArmor, 207–209
Groundwork hosts, 287
PROMPT option, 193
Prosecuting intruders, 233
ps command, 21–23

psql tool, 248
Pulling the plug, 233–234
PVs (physical volumes), 370
pwd command, 18
PXE boot server deployment. *See also* Installing Ubuntu Server; Kickstart; Preseeding.
  apache2 package, installing, 116
  boot prompts, responding to, 117
  DHCP server, setting up, 112–113
  mirroring the Ubuntu archive, 116
  netboot tarball, 113–115
  overview, 111
  Pxelinux, configuring, 113–115
  required services, 111–112
  required user interaction, 117
  server timeout value, setting, 117
  testing, 116–117
  TFTPD server, setting up, 113
  Web server, setting up, 116
PXE booting, 181
Pxelinux, configuring, 113–115
pxelinux menu, 118
python-software-properties package, 330

Q

—query argument, 352
Querying installed packages, 69–71
Question mark (?), wildcard character, 464
Quorum, 379

R

RAID (Redundant Array of Inexpensive Disks)
  as backup device, 241
  configuring after installation, 348–351
  configuring during installation, 346–348
  creating, 349–351
  current status, checking, 351–354
  fakeRAID, 344
  hardware, 344
  hardware/software hybrid. *See* fakeRAID.
  levels, 345
  migrating to. *See* Migrating to software RAID.
  minimum disk requirements, 345
  mirroring, 345
  partitioning disks, 346–349
  starting at boot time, 350
  striping, 345
  striping plus parity, 345
  UUID, specifying, 351
RAID (Redundant Array of Inexpensive Disks), software
  description, 344
  managing, 351–354
  migrating non-RAID disks to, 354–360
  /proc/mdstat file, 351–354
RAID 0, 345
RAID 1, 345
RAID 5. *See also* Migrating from RAID 1 to RAID 5.
  adding a drive to, 366–368
  description, 345
  reading with GRUB, 355
  as a root partition, 345
RAID MD (multidisk) devices, creating and using
  after installation, 349–350
  file system, 350, formatting
  during installation, 347–348
  mounting, 350
  number of active devices, specifying, 350
  partitions, specifying, 350
  RAID level, specifying, 350
RAID mdadm tool
  chaining commands, 353
  —create option, 350
  creating MD devices, 350–351
  —detail argument, 352
  —detail —scan command, 350–351
  disk failure, automatic notification, 353
  disks, setting as faulty, 353
  documentation, 354
  installing, 349
  —level option, 350
  —query argument, 352
  —raid-devices option, 350
  replacing a failed disk, 352–354
  resyncing swapped disks, 354
  software RAID management, 351–354
  swapping disks, 352–354
—raid-devices option, 350
RAM
  changing, 311–312
  copy of, 478

monitoring, 271
statistics, monitoring, 269–273
testing, 4, 427–428
usage, troubleshooting, 409–411
Random number generators, 478
Read permission, 472–473
reboot command, 39
Reboot the system, menu option, 438
Rebooting
immediate, disabling, 100
Ubuntu, 39
when troubleshooting, 403
Rebuilding packages, 77–79
reconfig command, 134
Recovery. *See* Rescue and recovery.
Redeploying the server, 234–235
Redirection
< (left angle bracket), redirection operator, 470–472
> (right angle bracket), redirection operator, 470–472
>> (right angle brackets), redirection operator, 470–472
to a black hole, 478
chaining operators, 472
of input, 471–472
to a null device, 478
of output, 470–472
overview, 470–472
overwriting the destination file, 471
Redundancy
disks. *See* RAID (Redundant Array of Inexpensive Disks).
fault tolerance, 342–343
LVM (Logical Volume Manager), 372
Redundant Array of Inexpensive Disks (RAID). *See* RAID (Redundant Array of Inexpensive Disks).
Regions *vs.* availability zones, 319
Regular expressions, in shell globs, 465–466
Reinstall GRUB boot loader, menu option, 437
relay_domains option, 153–154
relayhost option, 152
reload command
init scripts, 35
managing BIND, 134
managing Postfix, 148
System V init model, 29
Reloading configuration files
DNS servers, 134
Postfix, 148
services, 35
System V init model, 29
Reloading zone files, 134
Remote management, VMs (KVM), 313–315
Removable media, mount location, 44
—remove option, 353
Removing. *See also* Deleting.
duplicate lines in sorted output, 469–470
/etc/rc.boot, 228
MySQL databases, 166
packages, with Aptitude, 69
packages, with VMs (KVM), 306–307
passwords, 205
ufw rules, 219
Replacing failed disks, 352–354, 397
Reports directory, Tripwire, 232
Repositories. *See also* Package management.
adding, 72, 307
apt pinning, 75
backports, 74–75
cryptographic keys, 72–73
free software, 73
fully-supported software, 73
licensed software, 74
limitations, 74–75
listing, 64
main, 73
manipulating, 71–73
multiverse, 74
open source software, 74
partially-supported software, 73–74
pinning, 75
restricted, 73–74
Ubuntu defaults, 73–74
unintended updates, 75
universe, 74
reprepro project, 81
Rescue and recovery
help. *See* Help and resources.
resources. *See* Help and resources.
troubleshooting. *See* Troubleshooting.

Rescue and recovery, Ubuntu desktop live CD
booting from, 439
dd command, 445
ddrescue command, 445–447
deleted files, recovering, 440–443
fls tool, 441–443
forensic tools, 440–443
gpart tool, 443–444
Guess Partition tool, 443–444
hard drive rescue, 445–447
icat tool, 441–443
imaging drives, 445–447
imaging partitions, 446
partition table, restoring, 443–444
Sleuth Kit, 440–443
storing drive images, 445–447
universe repository, adding, 439–440
Rescue and recovery, Ubuntu recovery mode
clean option, 430
corrupted file systems, 432–433
disk space, freeing, 430
dpkg option, 431
file systems, 432–434
fsck tool, 432–433
fstab file mistakes, 433–434
GRUB, updating, 431
grub option, 431
mount command, 432
mounting file systems, 432–434
network option, 431
passwd command, 435
primary superblocks missing, 433
problems with init scripts, 434
recovery menu, 430–432
repairing packages, 431
resetting passwords, 435
resume option, 430
root option, 431
root shell, enabling, 431–432
system-summary option, 431
umount command, 432
unintentionally erasing file systems, 433
UUID, discovering, 434
UUID changed, 433–434
Rescue and recovery, Ubuntu server recovery CD
bad superblock, 439
booting into the CD, 436
Choose a different root file system, 438
Execute a shell in /dev/sdal, menu option, 436
Execute a shell in the installer environment, 437
GRUB recovery, 437–438
menu options, 436–438
overview, 435–436
Reboot the system, 438
Reinstall GRUB boot loader, 437
root file system repair, 438–439
Rescue disks, installation CDs as, 4
Reservations, LTSP IP addresses, 191–192
"Reserved blocks" field, 12
Resizing. *See* Sizing.
resolv.conf program, 47
Resource section, DRBD configuration file, 390–391
Resources. *See* Help and resources; Rescue and recovery; Troubleshooting.
respawn option, 384
Response time, fault tolerance, 343
restart command, 29, 35, 139–140
Restarting
Apache, 139–140
scripts, 35
restore command, 310
Restoring from backups. *See also* Backing up data; Rescue and recovery.
BackupPC file browser, 263–264
direct restore, 264
disk images, 243
download tar archive, 265
download zip archive, 264–265
mysql command, 246
MySQL databases, 246
options, 264–265
overview, 263–265
PostgreSQL databases, 248
Restoring from backups, file conventions
backup file directories, 266
configuration file directories, 265
/etc/backuppc, 265
/etc/backuppc/apache.conf, 265
/etc/backuppc/config.pl, 265
/etc/backuppc/hosts, 265
/etc/backuppc/htpasswd, 265
/etc/init.d/backuppc, 265
host definitions, 265
init script directory, 266

log file directory, 266
password directory, 265
/var/lib/backuppc, 266
/var/lib/backuppc/log, 266
/var/lib/backuppc/pc, 266
virtual host settings, 265
Restricted repositories, 73–74
resume command, 310–311
resume option, 430
Resuming KVM VMs, 310–311
Resyncing swapped disks, 354
retransfer zone command, 134
Retransferring zone files, 134
Right angle bracket (>), redirection operator, 470–472
Right angle brackets (>>), redirection operator, 470–472
rndc tool, 134
Rolling back to snapshots, 310
/root, editing, 228
Root account, enabling, 98
/root directory, 44
Root file system
mounting, 27
repairing, 438–439
Root kits, checking for, 236
root option, 431
Root partition size, specifying, 304
Root partitions, 6
Root shell, enabling, 431–432
Root squashing, disabling, 179
Root user
disk partitioning options, 7–13
home directory, 44
password, disabling, 109
permissions, assuming, 24
privileges, enabling, 109
rootpw command, 109
—rootsize option, 304
Round-robin policy, 373
route command, 49, 418–419
Routing outbound mail, 152
Routing to the remote host, troubleshooting
asterisks in the output, 422
closed ports *vs.* firewalls, 423–424
firewall rules, listing, 424–425
firewalls, detecting, 423–424
ICMP blocked, 423
listening ports, testing, 424
netstat command, 424
nmap command, 423–424
overview, 421–423
remote port, testing, 423–424
tcptraceroute package, 423
testing locally, 424
testing the route, 422–423
traceroute command, 422–423
ufw command, 424–425
RPM package format, 52
RRD files, 274
RSA keys, 160
rsync tweaks
backup retention, specifying, 262
blackout periods, 262–263
checksum-seed option, 258
excluding directories, 259–260
full backup interval, 261–262
FullAgeMax option, 262
FullKeepCnt option, 262
FullKeepCntMin option, 262
FullPeriod option, 261–262
host-specific tweaks, 260–261
limiting to one file system, 258–259
scheduling backups, 261–263
Rules. *See also* Firewalls, rules.
AppArmor, directory, 210
passwordless access to, 203
Rules, ufw program
delete command, 220
extended, 218–219
firewall, undoing, 217
iptables, rules directory, 225
removing, 219
syntax, 217–218
undoing, 220
Rules file, source packages, 62–63
Runlevels
default, changing, 33–34
System V init model, 28

S

safe-upgrade command, 65, 76
Samba file servers. *See also* NFS.
configuration, 176–177
configuration files, 175

Samba file servers, *continued*
databases used by, 176
DRBD, 396
/etc/init.d/nmdb, 175
/etc/init.d/smdb, 175
/etc/init/nmbd.conf, 175
/etc/init/smdb.conf, 175
/etc/samba/, 174
/etc/samba/smb.conf, 175
file conventions, 174–176
init script, 175
installing, 15, 174
Kickstart authentication, 109
log files, 176
overview, 174
passwords, setting, 175
sharing directories, 176–177
ufw firewall example, 223
user accounts, creating or disabling, 175
/usr/bin/smbpasswd, 175
/usr/share/doc/samba-doc/, 175
/var/lib/samba, 176
/var/log/samba/, 176
samba package, 15
samba-doc package, 15
sar tool, 270–273
Satellite system option, 146
save command, 310
/sbin directory, 40
/sbin/init program. *See* Boot process, /sbin/init program.
Scanning hard drives for problems, 444–447
Scheduling
database backups, 246–249
program execution. *See* at command; cron command.
system backups, 241, 261–263
Screen magnifier, enabling, 4
Screen reader, enabling, 4
Screen resolution, LTSP, 188
script option, 32
Scripts. *See* Init scripts.
Scrolling terminal output, 23
Search and replace text, 487–488
Search path missing, 421
Searching for
available packages, 65–67
files. *See* grep command.
installed packages, 69–71
Secondary Postfix servers, 153–154
secret-key variable, 332
Security
defense in depth, 201
encryption. *See* SSH security.
general principles, 200–201
greylisting mail servers, 154–156
intrusion detection. *See* IDSs (intrusion detection systems); Tripwire.
intrusion response. *See* Incident response.
keeping it simple, 200
layers of protection, 201
open relays, 153
permissions. *See* AppArmor; sudo command.
principle of least privilege, 200–201
responding to intrusion. *See* Incident response.
security by obscurity, 201
security patches, 201
servers. *See* OpenSSH servers; SSH security.
spam exposure, 153–156
X.509 certificates, 316–317
Security groups, Amazon EC2, 324–325
Selected time periods, monitoring, 272–273
Server BIOSs, 3
Server caches, flushing, 134
Server communication, troubleshooting
client connection, verifying, 416–417
client problems *vs.* server, 416–417
default gateway access, verifying, 418–419
dig tool, 420
DNS status, checking, 420–421
ethtool program, 417
ifconfig command, 418
inaccessible name server, 420–421
name server not configured, 420–421
name server problem, 420–421
network interface configuration, verifying, 418
nslookup command, 420–421
nslookup tool, 420
overview, 416
ping command, 419

route command, 418–419
search path missing, 421
Server roles
OpenSSH, 13
Server roles
DNS, 13
LAMP, 14
mail server, 14
OpenSSH, 13
packages, determining, 13
packages, listing, 15
PostgreSQL database, 14
print server, 14
Samba file server, 15
Tomcat Java server, 15
virtual machine host, 15
Servers
audible alarms, 487
databases. *See* MySQL databases; PostgreSQL databases.
deploying Web sites. *See* Web servers.
DNS services. *See* DNS servers.
dynamic host control. *See* DHCP servers.
Edubuntu. *See* LTSP (Linux Terminal Server Project).
for educational use. *See* LTSP (Linux Terminal Server Project).
e-mail. *See* Mail servers; POP/IMAP servers; Postfix mail server.
file. *See* File servers; NFS; Samba.
imaging, 234
killing, 380
redeploying after attack, 234–235
remote management. *See* OpenSSH servers.
SSH security settings, 211
Service checks
adding, 286, 291
creating, 294–295
critical thresholds, setting, 286
default settings, overriding, 293
options, changing, 286
warning thresholds, setting, 286
service command, 36
Service loading timeout, setting, 383
Service status, checking, 35
Services
booting. *See* Boot process services.
vs. commands, 288
deleting, 289
enabling with xinetd, 39
setmaxmem command, 311
setmem command, 311
Shell globs
? (question mark), wildcard character, 464
[ ] (square brackets), in regular expressions, 466
* (asterisk), wildcard character, 208–209, 465
character classes, 466
overview, 464–465
regular expressions, 465–466
Shell globs, pipes
| (vertical line), pipe symbol, 466–470
filtering input to, 467–468
grep command, 467–468
overview, 466–470
removing duplicate lines, 469–470
sort command, 468–469
sorting input to, 468–469
uniq command, 469–470
Shell globs, redirection
< (left angle bracket), redirection operator, 470–472
> (right angle bracket), redirection operator, 470–472
>> (right angle brackets), redirection operator, 470–472
to a black hole, 478
chaining operators, 472
of input, 471–472
of output, 470–472
overview, 470–472
overwriting the destination file, 471
Shell scripts, Kickstart, 110
Shooting the other node in the head, 380
+short option, 492–493
Shortcuts to
files. *See* Hard links; Symlinks.
pathnames, 486
show subcommand, 66–67
shutdown command, 310
Shutting down KVM VMs, 310

si: software interrupts, 407
Simplicity over complexity, troubleshooting, 401
Single points of failure, eliminating, 343
Site keys directory, 232
Size of files and directories, displaying, 20
Sizing partitions, 8, 10
skeleton script, 37
Slash (/), in IRC commands, 452
Slave server. *See* DNS slave server.
Slave zone files, location, 128
Sleuth Kit, 235–236, 440–443
sleuthkit package, 235–236
Smart upgrades, 59
smartctl tool, 426–427
smartd daemon, 269
Smartmontools, 268–269
smartmontools package, 426–427
smbfs package, 15
SMTP, 221–222
smtpd_recipient_restrictions option, 155
Snapshots. *See also* Backing up data; Suspending current state.
  KVM VMs, 310, 313
  rolling back to, 310
  taking, 310
Software. *See also specific software*.
  firewalls, 215
  interrupts, 407
Software RAID
  description, 344
  managing, 351–354
  migrating non-RAID disks to, 354–360
sort command, 468–469
Sorting input to pipes, 468–469
Sound in LTSP, 197
SOUND=True statement, 197
Source files, list of, 308
Source packages. *See* Package management, DEB format source packages.
sources.list file
  # (hash mark), comment indicator, 72
  manipulating repositories, 71–72
Spam exposure, 153–156
Specifying a distribution, 78
Split-brain policy, changing, 392
Split-brain problem, solving, 398
Split-brain syndrome, 379
Spool files, directory for, 43
Square brackets ([ ])
  in grep search keywords, 485–486
  in regular expressions, 466
ssh into Amazon EC2, 326–327
ssh key pairs, creating, 319–320
SSH keys
  configuring for BackupPC, 254
  copying, 308
  sharing, 491
  updating in LTSP, 195
SSH security. *See also* OpenSSH servers; Security; ufw program.
  botnets, 214
  brute-force attacks, 213–214
  configuration file, 211
  denyhosts program, 213–214
  failed logins, monitoring, 213–214
  firewalls, 214–216
  ipchains program, 215
  iptables program, 215
  keeping it simple, 215–216
  key-based authentication, 211–213
  overview, 210–211
  passphrases, 211–213
  password authentication, 211–213
  password prompt, bypassing, 213
  passwords, disabling, 213
  server settings, 211
  sshd_config file, 211
  TCP wrappers, hacking, 214
  thresholds, setting, 214
  ufw firewall example, 220–221
  whitelists for trusted hosts, 214
ssh-copy-id tool, 491
sshd_config file, 211
—ssh-key option, 308
—ssh-user-key option, 308
st: steal time, 407
start command, 29, 32, 139–140, 309–310
Starting/stopping
  Amazon EC2 instances, 327–328
  Apache, 139–140
  running processes, 21–23
  System V init model, 29

terminating Amazon EC2 instances, 327–328
Ubuntu, 39
Upstart jobs, 32
VMs (virtual machines), 309–310
Start-up scripts. *See* Init scripts; System V init model; Upstart.
Static configuration, DHCP servers, 162–163
Static IP address, assigning, 307
Static leases, 121–122
Status checking
Apache, 140–141
DNS servers, 134
extended-status command, 167
fullstatus command, 140–141
mail queues, 149
MySQL, 165–167
Postfix, 149
service option, 35
System V init model, 29
ufw program, 216
Upstart, 32
status command
Apache, 140–141
DNS servers, 134
MySQL, 167
Postfix, 149
service option, 35
System V init model, 29
ufw program, 216
Upstart, 32
Steal time, 407
stop command
Apache, 139–140
System V init model, 29
Upstart, 32
Stopping/starting. *See also* Killing processes.
Amazon EC2 instances, 327–328
Apache, 139–140
running processes, 21–23
System V init model, 29
terminating Amazon EC2 instances, 327–328
Ubuntu, 39
Upstart jobs, 32
VMs (virtual machines), 309–310
Storing drive images, 445–447
Striping plus parity, RAID disk, 345
Striping RAID disks, 345
Subnet mask, specifying, 307
sudo command. *See also* Security.
aliases, 205–206
assuming root permissions, 24
auto-expiration of access, 202
configuring for BackupPC, 255
features, 202–203
group-based access, 202
host-based access, 202
logging access, 203
password authentication, 202
passwordless access to rules, 203
superuser access, 202
sudo command, configuration file
% (percent sign), group name indicator, 204
changing, 203
checking for mistakes, 203
documentation, 204
location, 203
NOPASSWD: statement, 205
passwords, removing, 205
visudo tool, 203
Sudoers file
changing, 203
checking for mistakes, 203
sudo documentation, 204
—suite option, 303
Super user account, setting up, 169–170
Superblock problems, rescue and recovery, 439
Superuser access, sudo command, 202
Supplemental driver disk, Kickstart, 110
Support. *See* Help and resources; Rescue and recovery; Troubleshooting.
Support BIOS, enabling, 299
suspend command, 310–311
Suspending current state, KVM VMs, 310–311
Swap cache, monitoring, 271
swap option, 94
Swap partition size, specifying, 304
Swap space, 6
Swapping (physical disks), 352–354
Swapping (data in memory), troubleshooting, 411–412
—swapsize option, 304
Symbolic links. *See* Symlinks.

Symlinks
to Apache .load and .conf files, 137–138
creating, 38, 475
directories, 20
files, 20
identifying, 20, 475
overview, 475
uses for, 475
to virtual hosts, 138
Synaptic, 64–65
Syncing arrays
migrating from RAID 1 to RAID 5, 366
migrating non-RAID disks to RAID, 358
/sys directory, 45
Syslog facility, 384
Syslog file example, 387–388
sysstat package, 269–273, 411–412
System administration. *See* Command-line administration; Managing.
System administrator
disk partitioning options, 7–13
home directory, 44
password, disabling, 109
permissions, assuming, 24
privileges, enabling, 109
System configuration files, directory for, 42–43
System libraries, directory for, 41
System load
average, 404–406
monitoring, 269–273
System logs, directory for, 43
System time, displaying, 38
System V init model. *See also* Init scripts; Upstart.
description, 27–28
drawbacks, 30–31
/etc/init.d script, 28–29
/etc/rc0.d — /etc/rc06.d scripts, 29
/etc/rc.local script, 29
/etc/rcS.d script, 29
force-reload command, 29
init scripts, 28–29
reload command, 29
reloading settings, 29
restart command, 29
runlevels, 28
start command, 29
starting/stopping, 29
start-up scripts, 28–30
status command, 29
stop command, 29
user scripts, 29
system-config-kickstart package, installing, 105
system-summary option, 431

T

t1.micro instances, 315
Tar archives, restoring from, 265
tasksel command, 15
TCP wrappers, hacking, 214
tcpdump program, 276
tcptraceroute package, 423
Technical support. *See* Help and resources; Rescue and recovery; Troubleshooting.
Telnet, testing e-mail, 490–491
Temporary files, partitions for, 7
Terminating Amazon EC2 instances, 327–328
Testing
backups, 241
fail-over, 378
fault tolerance, 343
hard drives, 426–427
listening ports, 424
memory, 4
PXE boot server deployment, 116–117
RAM, 4, 427–428
remote ports, 423–424
routing to the remote host, 422–424
Text in files, search and replace, 487–488
TFTPD (Trivial File Transfer Protocol Daemon), 39
TFTPD server, setting up, 113
Thin clients, 183
Third-party programs
directory for, 42
partitions for, 6
Thresholds for SSH security, setting, 214
Time between heartbeats, setting, 383
Timeout
DHCP duration, setting, 90
PXE boot servers, 117
service loading, setting, 383
TIMEOUT option, 193
TIMEOUT value, setting, 117

/tmp directory, 7, 45
tmp files, excessive disk space, 415
Tomcat Java server, 15
Tools for package management, 64–67, 69
top command, 406–408, 410
tps: transfers per second, 412
+trace option, 492–493
traceroute command, 422–423
Tripwire
  configuration files directory, 232
  encrypted settings, directory for, 232
  /etc/rc.boot, removing, 228
  /etc/tripwire/, 232
  /etc/tripwire/*-local.key, 232
  /etc/tripwire/*-site.key, 232
  /etc/tripwire/tw.cfg, 232
  /etc/tripwire/twcfg.txt, 232
  /etc/tripwire/tw.pol, 232
  /etc/tripwire/twpol.txt, 227, 232
  file conventions, 227–228, 232–233
  keys, 227, 232
  local keys directory, 232
  log directory, 233
  MTA (Mail Transport Agent), 227
  passphrases, 227
  policies, updating, 227–228
  policy file, editing, 227–228
  policy files directory, 232
  /proc, editing, 228
  reports directory, 232
  /root, editing, 228
  site keys directory, 232
  /var/lib/tripwire/, 232
  /var/lib/tripwire/reports, 232
  /var/log/syslog, 233
Tripwire database
  default directory, 232
  "file does not exist" message, 229
  initializing, 229–230
  "unknown file system type" message, 229
  updating, 230–231
Trivial File Transfer Protocol Daemon (TFTPD), 39
Troubleshooting. *See also* Help and resources; Rescue and recovery.
  bash commands, too many arguments, 488–489
  checking installation CDs for defects, 4
  DHCP timing out, 90
  error retrieving preseed.cfg file, 89
  "file does not exist" message, Tripwire, 229
  file system will not unmount, 490
  "ifconfig: command not found" message, 49
  "unknown file system type" message, Tripwire, 229
  "Wrong file system type" message, 179
Troubleshooting, general principles
  communicating with collaborators, 402
  dividing the problem space, 400–401
  documenting problems and solutions, 402–403
  favoring past solutions, 401–402
  Internet as reference, 403
  resisting rebooting, 403
  simplicity over complexity, 401
  understanding the system, 402
Troubleshooting, hardware
  hard drives, testing, 426–427
  ifconfig command, 49, 425–426
  Memtest86+ tool, 427–428
  network card errors, 425–426
  RAM, testing, 427–428
  smartctl tool, 426–427
  smartmontools package, 426–427
Troubleshooting, localhost sluggish or unresponsive
  Blk_read: total blocks read, 412
  Blk_read/s: blocks read per second, 412
  Blk_wrtn: total blocks written, 412
  Blk_wrtn/s: blocks written per second, 412
  excessive swapping, 411–412
  hi: hardware interrupts, 407
  high I/O wait, 411–412
  high user time, 408–409
  id: CPU idle time, 407
  iostat program, 411–412
  iotop program, 413
  metrics, 404–407
  ni: nice CPU time, 407
  OOM (out-of-memory) killer, 410–411
  out-of-memory issues, 409–411
  overview, 404
  RAM usage, 409–411
  si: software interrupts, 407
  st: steal time, 407

Troubleshooting, localhost sluggish or unresponsive, *continued*
sy: system CPU time, 407
sysstat package, 411–412
system load average, 404–406
top command, 406–408
tps: transfers per second, 412
uptime command, 404–405
us: user CPU time, 407
wa: I/O wait, 407
Troubleshooting, out of disk space
df command, 413–416
du command, 414–415
duck command, 414–415
excessive tmp files, 415
full file system, 415
out of inodes, 415–416
usage, by directory, 414–415
usage, by file system, 413–415
Troubleshooting networks, routing to the remote host
asterisks in the output, 422
closed ports *vs.* firewalls, 423–424
firewall rules, listing, 424–425
firewalls, detecting, 423–424
ICMP blocked, 423
listening ports, testing, 424
netstat command, 424
nmap command, 423–424
remote port, testing, 423–424
tcptraceroute package, 423
testing locally, 424
testing the route, 422–423
traceroute command, 422–423
ufw command, 424–425
Troubleshooting networks, servers can't communicate
client connection, verifying, 416–417
client problems *vs.* server, 416–417
default gateway access, verifying, 418–419
dig tool, 420
DNS status, checking, 420–421
ethtool program, 417
inaccessible name server, 420–421
name server not configured, 420–421
name server problem, 420–421
network interface configuration, verifying, 418
nslookup command, 420–421
overview, 416
ping command, 419
route command, 418–419
search path missing, 421
Troubleshooting networks, slow network speeds, 417–418
"Typical usage" field, 12

U

Ubuntu
desktop live CD. *See* Rescue and recovery, Ubuntu desktop live CD.
recovery mode. *See* Rescue and recovery, Ubuntu recovery mode.
server recovery CD. *See* Rescue and recovery, Ubuntu server recovery CD.
version, specifying, 303
#ubuntu chat, 452
#ubuntu-server chat, 452
ubuntu-vm-builder. *See* vmbuilder tool.
ubuntu-vm-builder script, installing, 299. *See also* vmbuilder tool; VMs (KVM), creating.
ufw command, 424–425
ufw program. *See also* SSH security.
configuration file directory, 224–225
environment variables directory, 225
/etc/defaults/ufw, 225
/etc/init.d/ufw, 225
/etc/ufw/, 224–225
/etc/ufw/after6.rules, 225
/etc/ufw/after.rules, 225
/etc/ufw/before6.rules, 225
/etc/ufw/before.rules, 225
file conventions, 224–226
init script directory, 225
iptables, rules directory, 225
/lib/ufw/user6.rules, 225
/lib/ufw/user.rules, 225
/var/log/syslog, 226
ufw program, firewall examples
DNS, 221
MySQL, 222
NFS, 223–224
POP/IMAP, 222

PostgreSQL, 223
Samba, 223
SMTP, 221–222
SSH, 220–221
Web, 221
ufw program, firewalls
allow command, 217
default command, 216–217
default policy, defining, 216–217
delete allow command, 217
delete deny command, 217
deny command, 217
disable command, 216
enable command, 216
enabling/disabling, 216
extended rules, 218–219
installing, 216
locking yourself out, 220
logging command, 217
logs, dumping, 217
periodic disabling, 220
remote management, 220
rule syntax, 217–218
rules, denying by default, 217
rules, undoing, 217
status, checking, 216
status command, 216
undoing rules, 220
UID default, 98
umount command
file system rescue and recovery, 432
file system will not unmount, 490
Understanding the system, troubleshooting, 402
Uninstalling packages, 57, 59, 69
uniq command, 469–470
Universe repositories, 74, 439–440
Unpacking source packages, 62
update-grub program, 25
Update-initramfs argument, changing, 362
update-rc.d program, 36
Updating aliases, 150
Upgrade command, 65
Upgrading
automating, 76
do-release-upgrade program, 76
full-system, 75–76
packages, automatically, 65
shared libraries, 57
smart, 59
to unintended versions, 75
Upstart. *See also* System V init model.
(hash mark), comment indicator, 31
checking job status, 32–33
comments, 31
default runlevel, changing, 33–34
description, 30–31
event-driven actions, 30
exec option, 32
initctl command, 32–33
job status, querying, 32–33
script location, 31
script option, 32
script syntax, 31
start command, 32
starting/stopping jobs, 32
status command, 32
stop command, 32
uptime command, 404–405
us: user CPU time, 407
USB devices, LTSP, 195–196
"Use as" field, 10–11
User accounts. *See also* Root user.
default, disabling, 99
default, specifying, 304
PostgreSQL, creating/deleting, 169–170
Samba, creating/disabling, 175
user command, 109
User CPU time, 407
User mailbox directory, 147
—user option, 304
User scripts, System V init model, 30
User time too high, troubleshooting, 408–409
Usernames, mapping to PostgreSQL usernames, 171
Users
alias mappings, 147, 150
AMIs, default, 327
group membership, displaying, 473
KVM, setting up, 299
LTSP, creating first, 188
Users, initial settings
Kickstart, 109
preseeding, 98–99

/usr directory, 7, 41
usr/bin directory, 41
/usr/bin/smbpasswd, 175
/usr/lib directory, 41
/usr/lib/cgi-bin/, 139
/usr/local directory, 41
/usr/sbin directory, 41
/usr/share/doc/samba-doc/, 175
UUID
  changed, rescue and recovery, 433–434
  discovering, 434
  migrating non-RAID disks to RAID, 358
  specifying for RAID disks, 351
uupdate program, 79–80

V

/var directory, 6, 43
/var/cache/bind, 128
Variable-size data, partitioning for, 6
Variable-size files, directory for, 43
/var/lib/backuppc, 266
/var/lib/backuppc/log, 266
/var/lib/backuppc/pc, 266
/var/lib/dhcp3/dhcpd.leases, 161
/var/lib/mysql/, 165
/var/lib/samba, 176
/var/lib/tripwire/, 232
/var/lib/tripwire/reports, 232
/var/log directory, 43
/var/log/apache2/, 139
/var/log/apparmor/, 210
/var/log/auth.log, 160
/var/log/mail.*, 147–148
/var/log/mail.log, 158
/var/log/postgresql/, 171
/var/log/samba/, 176
/var/log/syslog
  AppArmor, 210
  DHCP servers, 161
  DNS servers, BIND, 129
  Dovecot, 158
  MySQL databases, 165
  NFS, 178
  SSH security, ufw program, 226
  Tripwire, 233
/var/spool directory, 43
/var/spool/mail/, 147
/var/spool/postfix/, 147
/var/www/, 139
/var/www directory, 43
Vertical line (|), pipe symbol, 466–470
VGs (volume groups), 370
Vhost_alias module, 142–143
virsh command, 299–300, 309–312
virt-manager utility, 312–315
Virtual file systems, directory for, 45
Virtual host settings, BackupPC, 265
Virtual hosts, Apache Web server, 138
Virtual machines. *See* VMs.
VirtualDocumentRoot option, 142–143
Virtualization. *See also* Amazon EC2; Juju; VMs.
  extensions, confirming, 298
  overview, 298
  technologies. *See* KVM.
  virtual machine host server, 15
Visually impaired users. *See* Accessibility options.
visudo tool, 203
vmbuilder tool, 302–309
VMs (KVM), creating. *See also* KVM.
  adding new VM to local KVM instance, 304
  —arch option, 304
  automating, 308–309
  —copy option, 308
  destination directory, specifying, 304
  destination files, list of, 308
  —execscript option, 308
  firstboot option, 309
  firstlogin option, 309
  —flavour option, 303–304
  hostname, specifying, 304
  —hostname option, 304
  JeOS, 302
  kernel flavor, specifying, 303–304
  —libvirt option, 304
  —part option, 306
  —pass option, 304
  —rootsize option, 304
  —suite option, 303
  —swapsize option, 304
  —user option, 304

VMs (KVM), managing
autostart command, 310
current load, 313–315
destroy command, 310
graphical console, 313–314
hardware, 313–314
power off, 310
RAM, changing, 311–312
remote management, 313–315
restore command, 310
resume command, 310–311
resuming, 310–311
rolling back to snapshots, 310
save command, 310
setmaxmem command, 311
setmem command, 311
shutdown command, 310
shutting down, 310
snapshotting, 310, 313
start command, 309–310
starting at boot time, 310
starting the VM, 309–310
suspend command, 310–311
suspending current state, 310–311. *See also* Snapshots.
virsh command, 309–312
virt-manager utility, 312–315
VMs (KVM), networking
—bcast option, 308
broadcast address, specifying, 308
creating, 302–305
DNS address, specifying, 308
—dns option, 308
domain default, specifying, 307
domain option, 307
gateway address, specifying, 308
—gw, 308
host network address, specifying, 307
—ip option, 307
—mask option, 307
—net option, 307
overview, 300–302
partitioning disks, 306
password default, specifying, 304
post-install scripts, 308–309
processor architecture, specifying, 304
root partition size, specifying, 304
sample command, 303
sample network, 305
source files, list of, 308
SSH key files, copying, 308
—ssh-key option, 308
—ssh-user-key option, 308
static IP address, assigning, 307
subnet mask, specifying, 307
swap partition size, specifying, 304
Ubuntu version, specifying, 303
user default, specifying, 304
vmbuilder tool, 302–309
VMs (KVM), package management
adding packages, 306–307
—addpkg option, 306–307
—components option, 307
—mirror option, 307
—ppa option, 307
PPAs (Personal Package Archives), 307
—removepkg option, 306–307
removing packages, 306–307
repositories, adding, 307
Ubuntu mirror, specifying, 307
VMs (KVM), scripts
interactive, 309
running on command, 308
running on first VM boot, 309

W

wa: I/O wait, 407
Warning prompts for preseeding partitioning, disabling, 91
Warning thresholds, setting, 286
warntime option, 383
watch command, 487
Web forums, 451
Web servers. *See also* Apache.
installing, 135–136
LAMP environment, 135, 141–144
setting up for PXE boot server deployment, 116
Web service, ufw firewall example, 221
Web-based administration, PostgreSQL, 171–173
which command, 486

Whitelists
for trusted hosts, 214
tweaking, 155–156
Wildcard characters
? (question mark), wildcard character, 464
* (asterisk), in shell globs, 208–209, 465
in the exec command, 488
in the find command, 488
winbind package, 15
Wireless adapters, bridging support, 302
WordPress, 141–144
Write permission, 472–473

X

X.509 certificates, 316–317
xargs program, 489
XChat program, 452–455
xconfig—monitor option, 110
xinetd
description, 38–39
echo feature, 38
enabling services, 39
FTP feature, 39
managing services, 38–39
system time, displaying, 38
TFTPD (Trivial File Transfer Protocol Daemon), 39
XOR policy, 373

Z

Zip archives, restoring from, 264–265
Zone files
adding, 129–132
location, 128
ownership, 131
permissions, 131
referencing in named.conf, 131–132
reloading, 134
retransferring, 134