O'REILLY®

Second Edition

Linux Cookbook

Essential Skills for Linux Users and System & Network Administrators



Carla Schroder

O'REILLY®

Linux Cookbook

This handy cookbook teaches new-to-intermediate Linux users the essential skills necessary to manage a Linux system, using both graphical and command-line tools. Whether you run Linux in embedded, desktop, server, or cloud or virtual environments, the fundamental skills are the same. This book aims to get you up and running quickly, with copy-paste examples.

Carla Schroder provides recipes that cover specific problems, with discussions that explain how each recipe works, as well as references for additional study.

You'll learn how to:

- Use systemd, the new comprehensive service manager
- Build simple or complex firewalls with firewalld
- Set up secure network connections for Linux systems and mobile devices
- Rescue nonbooting systems
- Reset lost passwords on Linux and Windows
- Use dnsmasq to simplify managing your LAN name services
- Manage users and groups and control access to files
- Probe your computer hardware and monitor hardware health
- Manage the GRUB bootloader and multiboot Linux and Windows
- Keep accurate time across your network with the newest tools
- Build an internet router/firewall on Raspberry Pi
- Manage filesystems and partitioning

"A must-read for learning Linux. Carla Schroder walks you through every aspect of the Linux operating system in such a way that anyone can follow."

—**Jack Wallen**
Award-winning tech writer for *TechRepublic*, *The New Stack*, and more

"Highly effective engineers know their tools and how to use them. Carla Schroder's expertise opens your eyes wider to those things in Linux that you didn't know you didn't know."

—**Jonathan Johnson**
Independent software consultant and trainer, Dijure

Carla Schroder is a tech journalist and technical writer with experience as a system and network administrator running mixed Linux-Microsoft-Apple networks. She's written over 1,000 Linux how-tos for various publications and currently writes and maintains product manuals for a Linux enterprise software company. Carla is also the author of *Linux Networking Cookbook* and *The Book of Audacity*.

LINUX

US $59.99 CAN $79.99

ISBN: 978-1-492-08716-8



Twitter: @oreillymedia
facebook.com/oreilly

SECOND EDITION

Linux Cookbook

Essential Skills for Linux Users and System and Network Administrators

Carla Schroder

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Linux Cookbook
by Carla Schroder

Copyright © 2021 Carla Schroder. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Suzanne McQuade
Development Editor: Jeff Bleiel
Production Editor: Daniel Elfanbaum
Copyeditor: Sonia Saruba
Proofreader: Tom Sullivan

Indexer: nSight, Inc.
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

December 2004: First Edition
September 2021: Second Edition

Revision History for the Second Edition
2021-08-12: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781492087168* for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Linux Cookbook*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

The views expressed in this work are those of the author, and do not represent the publisher's views. While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-492-08716-8

[LSI]

Table of Contents

Preface xiii

1. Installing Linux 1
Booting from Installation Media 2
Where to Download Linux 3
Best Linux for Newbies 3
1.1 Entering your System BIOS/UEFI Setup 4
1.2 Downloading a Linux Installation Image 6
1.3 Creating a Linux Installation USB Stick with UNetbootin 7
1.4 Creating a Linux Installation DVD with K3b 9
1.5 Using the wodim Command to Create a Bootable CD/DVD 12
1.6 Creating a Linux Installation USB Stick with the dd Command 13
1.7 Trying a Simple Ubuntu Installation 15
1.8 Customizing Partitioning 18
1.9 Preserving Existing Partitions 22
1.10 Customizing Package Selection 23
1.11 Multibooting Linux Distributions 29
1.12 Dual-boot with Microsoft Windows 31
1.13 Recovering an OEM Windows 8 or 10 Product Key 34
1.14 Mounting Your ISO Image on Linux 35

2. Managing the GRUB Bootloader 37
2.1 Rebuilding Your GRUB Configuration File 40
2.2 Unhiding a Hidden GRUB Menu 40
2.3 Booting to a Different Linux Kernel 41
2.4 Understanding GRUB Configuration Files 43
2.5 Writing a Minimal GRUB Configuration File 44
2.6 Setting a Custom Background for Your GRUB Menu 48

2.7 Changing Font Colors in the GRUB Menu 49
2.8 Applying a Theme to Your GRUB Menu 52
2.9 Rescuing a Nonbooting System from the grub> Prompt 54
2.10 Rescuing a Nonbooting System from the grub rescue> Prompt 56
2.11 Reinstalling Your GRUB Configuration 58

3. Starting, Stopping, Restarting, and Putting Linux into Sleep Modes. 59
3.1 Shutting Down with systemctl 60
3.2 Shutting Down, Timed Shutdowns, and Rebooting with the shutdown Command 61
3.3 Shutting Down and Rebooting with halt, reboot, and poweroff 63
3.4 Sending Your System into Sleep Modes with systemctl 64
3.5 Rebooting Out of Trouble with Ctrl-Alt-Delete 66
3.6 Disabling, Enabling, and Configuring Ctrl-Alt-Delete in the Linux Console 68
3.7 Creating Scheduled Shutdowns with cron 69
3.8 Scheduling Automated Startups with UEFI Wake-Ups 71
3.9 Scheduling Automated Startups with RTC Wake-ups 73
3.10 Setting Up Remote Wake-Ups with Wake-on-LAN over Wired Ethernet 75
3.11 Setting Up Remote Wake-Ups over WiFi (WoWLAN) 77

4. Managing Services with systemd. 79
4.1 Learning if Your Linux Uses systemd 82
4.2 Understanding PID 1, the Mother of All Processes 84
4.3 Listing Services and Their States with systemctl 86
4.4 Querying the Status of Selected Services 89
4.5 Starting and Stopping Services 91
4.6 Enabling and Disabling Services 92
4.7 Stopping Troublesome Processes 94
4.8 Managing Runlevels with systemd 95
4.9 Diagnosing Slow Startups 98

5. Managing Users and Groups. 99
5.1 Finding a User's UID and GID 101
5.2 Creating a Human User with useradd 103
5.3 Creating a System User with useradd 105
5.4 Changing the useradd Default Settings 106
5.5 Customizing the Documents, Music, Video, Pictures, and Downloads Directories 108
5.6 Creating User and System Groups with groupadd 110
5.7 Adding Users to Groups with usermod 112
5.8 Creating Users with adduser on Ubuntu 113

5.9 Creating a System User with adduser on Ubuntu 114
5.10 Creating User and System Groups with addgroup 115
5.11 Checking Password File Integrity 116
5.12 Disabling a User Account 117
5.13 Deleting a User with userdel 118
5.14 Deleting a User with deluser on Ubuntu 119
5.15 Removing a Group with delgroup on Ubuntu 120
5.16 Finding and Managing All Files for a User 120
5.17 Using su to Be Root 122
5.18 Granting Limited Root Powers with sudo 123
5.19 Extending the sudo Password Timeout 126
5.20 Creating Individual sudoers Configurations 127
5.21 Managing the Root User's Password 127
5.22 Changing sudo to Not Ask for the Root Password 128

6. Managing Files and Directories. 131
6.1 Creating Files and Directories 133
6.2 Quickly Creating a Batch of Files for Testing 134
6.3 Working with Relative and Absolute Filepaths 136
6.4 Deleting Files and Directories 137
6.5 Copying, Moving, and Renaming Files and Directories 139
6.6 Setting File Permissions with chmod's Octal Notation 140
6.7 Setting Directory Permissions with chmod's Octal Notation 142
6.8 Using the Special Modes for Special Use Cases 143
6.9 Removing the Special Modes in Octal Notation 146
6.10 Setting File Permissions with chmod's Symbolic Notation 146
6.11 Setting the Special Modes with chmod's Symbolic Notation 148
6.12 Setting Permissions in Batches with chmod 150
6.13 Setting File and Directory Ownership with chown 151
6.14 Changing Ownership on Batches of Files with chown 152
6.15 Setting Default Permissions with umask 153
6.16 Creating Shortcuts (Soft and Hard Links) to Files and Directories 154
6.17 Hiding Files and Directories 157

7. Backup and Recovery with rsync and cp. 159
7.1 Selecting Which Files to Back Up 161
7.2 Selecting Files to Restore from Backups 162
7.3 Using the Simplest Local Backup Method 163
7.4 Automating Simple Local Backups 164
7.5 Using rsync for Local Backups 166
7.6 Making Secure Remote File Transfers with rsync over SSH 168
7.7 Automating rsync Transfers with cron and SSH 170

7.8 Excluding Files from Backup 170
7.9 Including Selected Files to Backup 172
7.10 Managing Includes with a Simple Include File 173
7.11 Managing Includes and Excludes with an Exclude File 174
7.12 Limiting rsync's Bandwidth Use 176
7.13 Building an rsyncd Backup Server 177
7.14 Limiting Access to rsyncd Modules 180
7.15 Creating a Message of the Day for rsyncd 182

8. Managing Disk Partitioning with parted. 185
Overview 185
8.1 Unmounting Your Partitions Before Using parted 190
8.2 Choosing the Command Mode for parted 191
8.3 Viewing Your Existing Disks and Partitions 192
8.4 Creating GPT Partitions on a Nonbooting Disk 195
8.5 Creating Partitions for Installing Linux 197
8.6 Removing Partitions 198
8.7 Recovering a Deleted Partition 199
8.8 Increasing Partition Size 200
8.9 Shrinking a Partition 202

9. Managing Partitions and Filesystems with GParted. 205
9.1 Viewing Partitions, Filesystems, and Free Space 207
9.2 Creating a New Partition Table 209
9.3 Deleting a Partition 210
9.4 Creating a New Partition 211
9.5 Deleting a Filesystem Without Deleting the Partition 213
9.6 Recovering a Deleted Partition 214
9.7 Resizing Partitions 215
9.8 Moving a Partition 216
9.9 Copying a Partition 218
9.10 Managing Filesystems with GParted 220

10. Getting Detailed Information About Your Computer Hardware. 223
10.1 Collecting Hardware Information with lshw 224
10.2 Filtering lshw Output 226
10.3 Detecting Hardware, Including Displays and RAID Devices, with hwinfo 227
10.4 Detecting PCI Hardware with lspci 228
10.5 Understanding lspci Output 230
10.6 Filtering lspci Output 231
10.7 Using lspci to Identify Kernel Modules 234
10.8 Using lsusb to List USB Devices 235

10.9 Listing Partitions and Hard Disks with lsblk 237
10.10 Getting CPU Information 238
10.11 Identifying Your Hardware Architecture 240

11. Creating and Managing Filesystems. 243
Filesystem Overview 244
11.1 Listing Supported Filesystems 246
11.2 Identifying Your Existing Filesystems 248
11.3 Resizing Filesystems 249
11.4 Deleting Filesystems 250
11.5 Using a New Filesystem 251
11.6 Creating Automatic Filesystem Mounts 253
11.7 Creating Ext4 Filesystems 256
11.8 Configuring the Ext4 Journal Mode 257
11.9 Finding Which Journal Your Ext4 Filesystem Is Attached To 259
11.10 Improving Performance with an External Journal for Ext4 260
11.11 Freeing Space from Reserved Blocks on Ext4 Filesystems 262
11.12 Creating a New XFS Filesystem 263
11.13 Resizing an XFS Filesystem 264
11.14 Creating an exFAT Filesystem 266
11.15 Creating FAT16 and FAT32 Filesystems 267
11.16 Creating a Btrfs Filesystem 269

12. Secure Remote Access with OpenSSH. 273
12.1 Installing OpenSSH Server 275
12.2 Generating New Host Keys 276
12.3 Configuring Your OpenSSH Server 276
12.4 Checking Configuration Syntax 279
12.5 Setting Up Password Authentication 279
12.6 Retrieving a Key Fingerprint 281
12.7 Using Public Key Authentication 282
12.8 Managing Multiple Public Keys 284
12.9 Changing a Passphrase 285
12.10 Automatic Passphrase Management with Keychain 286
12.11 Using Keychain to Make Passphrases Available to Cron 287
12.12 Tunneling an X Session Securely over SSH 288
12.13 Opening an SSH Session and Running a Command in One Line 290
12.14 Mounting Entire Remote Filesystems with sshfs 291
12.15 Customizing the Bash Prompt for SSH 292
12.16 Listing Supported Encryption Algorithms 294

13. Secure Remote Access with OpenVPN. 297
OpenVPN Overview 297
13.1 Installing OpenVPN, Server and Client 299
13.2 Setting Up a Simple Connection Test 300
13.3 Setting Up Easy Encryption with Static Keys 302
13.4 Installing EasyRSA to Manage Your PKI 304
13.5 Creating a PKI 306
13.6 Customizing EasyRSA Default Options 311
13.7 Creating and Testing Server and Client Configurations 312
13.8 Controlling OpenVPN with systemctl 315
13.9 Distributing Client Configurations More Easily with .ovpn Files 316
13.10 Hardening Your OpenVPN Server 320
13.11 Configuring Networking 323

14. Building a Linux Firewall with firewalld. 325
firewalld Overview 325
14.1 Querying Which Firewall Is Running 328
14.2 Installing firewalld 330
14.3 Finding Your firewalld Version 331
14.4 Configuring iptables or nftables as the firewalld Backend 332
14.5 Listing All Zones and All Services Managed by Each Zone 332
14.6 Listing and Querying Services 335
14.7 Selecting and Setting Zones 336
14.8 Changing the Default firewalld Zone 338
14.9 Customizing firewalld Zones 339
14.10 Creating a New Zone 340
14.11 Integrating NetworkManager and firewalld 342
14.12 Allowing or Blocking Specific Ports 343
14.13 Blocking IP Addresses with Rich Rules 345
14.14 Changing a Zone Default Target 346

15. Printing on Linux. 347
Overview 347
15.1 Using the CUPS Web Interface 350
15.2 Installing a Locally Attached Printer 350
15.3 Giving Printers Useful Names 354
15.4 Installing a Network Printer 355
15.5 Using Driverless Printing 357
15.6 Sharing Nonnetworked Printers 359
15.7 Correcting the “Forbidden” Error Message 360
15.8 Installing Printer Drivers 362
15.9 Modifying an Installed Printer 364

15.10 Saving Documents by Printing to a PDF File 365
15.11 Troubleshooting 366

16. Managing Local Name Services with Dnsmasq and the hosts File. 367
16.1 Simple Name Resolution with /etc/hosts 368
16.2 Using /etc/hosts for Testing and Blocking Annoyances 371
16.3 Finding All DNS and DHCP Servers on Your Network 372
16.4 Installing Dnsmasq 374
16.5 Making systemd-resolved and NetworkManager Play Nice with Dnsmasq 375
16.6 Configuring Dnsmasq for LAN DNS 376
16.7 Configuring firewalld to Allow DNS and DHCP 379
16.8 Testing Your Dnsmasq Server from a Client Machine 380
16.9 Managing DHCP with Dnsmasq 381
16.10 Advertising Important Services over DHCP 383
16.11 Creating DHCP Zones for Subnets 384
16.12 Assigning Static IP Addresses from DHCP 385
16.13 Configuring DHCP Clients for Automatic DNS Entries 386
16.14 Managing Dnsmasq Logging 388
16.15 Configuring Wildcard Domains 389

17. Keeping Time with ntpd, chrony, and timesyncd. 391
17.1 Finding Which NTP Client Is on Your Linux System 392
17.2 Using timesyncd for Simple Time Synchronization 394
17.3 Setting Time Manually with timedatectl 396
17.4 Using chrony for Your NTP Client 397
17.5 Using chrony as a LAN Time Server 398
17.6 Viewing chrony Statistics 400
17.7 Using ntpd for Your NTP Client 401
17.8 Using ntpd for Your NTP Server 403
17.9 Managing Time Zones with timedatectl 404
17.10 Managing Time Zones Without timedatectl 405

18. Building an Internet Firewall/Router on Raspberry Pi. 407
Overview 407
18.1 Starting and Shutting Down Raspberry Pi 410
18.2 Finding Hardware and How-Tos 411
18.3 Cooling the Raspberry Pi 413
18.4 Installing Raspberry Pi OS with Imager and dd 413
18.5 Installing Raspberry Pi with NOOBS 415
18.6 Connecting to a Video Display Without HDMI 417
18.7 Booting into Recovery Mode 420

18.8 Adding a Second Ethernet Interface 420
18.9 Setting Up an Internet Connection Sharing Firewall with firewalld 424
18.10 Running Your Raspberry Pi Headless 427
18.11 Building a DNS/DHCP Server with Raspberry Pi 428

19. System Rescue and Recovery with SystemRescue. 431
19.1 Creating Your SystemRescue Bootable Device 432
19.2 Getting Started with SystemRescue 432
19.3 Understanding SystemRescue's Two Boot Screens 434
19.4 Understanding SystemRescue's Boot Options 437
19.5 Identifying Filesystems 438
19.6 Resetting a Linux Root Password 439
19.7 Enabling SSH in SystemRescue 440
19.8 Copying Files over the Network with scp and sshfs 442
19.9 Repairing GRUB from SystemRescue 445
19.10 Resetting a Windows Password 446
19.11 Rescuing a Failing Hard Disk with GNU ddrescue 448
19.12 Managing Partitions and Filesystems from SystemRescue 450
19.13 Creating a Data Partition on Your SystemRescue USB Drive 451
19.14 Preserving Changes in SystemRescue 453

20. Troubleshooting a Linux PC. 455
Overview 455
20.1 Finding Useful Information in Logfiles 457
20.2 Configuring journald 461
20.3 Building a Logging Server with systemd 462
20.4 Monitoring Temperatures, Fans, and Voltages with lm-sensors 465
20.5 Adding a Graphical Interface to lm-sensors 467
20.6 Monitoring Hard Disk Health with smartmontools 470
20.7 Configuring smartmontools to Send Email Reports 473
20.8 Diagnosing a Sluggish System with top 475
20.9 Viewing Selected Processes in top 477
20.10 Escaping from a Frozen Graphical Desktop 478
20.11 Troubleshooting Hardware 479

21. Troubleshooting Networks. 481
Diagnostic Hardware 481
21.1 Testing Connectivity with ping 482
21.2 Profiling Your Network with fping and nmap 484
21.3 Finding Duplicate IP Addresses with arping 487
21.4 Testing HTTP Throughput and Latency with httping 488
21.5 Using mtr to Find Troublesome Routers 490

Appendix. Software Management Cheatsheets **493**

Index **501**

Preface

Way back in olden times I wrote the first edition of the *Linux Cookbook*, which was released unto a joyous world in 2004. It sold well, I heard from many happy readers, and some are still my friends.

For a Linux book, 17 years old is ancient. Linux was 14 years old in 2004, a wee baby computer operating system. Even so, it was already a popular and widely used powerhouse, adapting to any role, from tiny embedded devices to mainframes and supercomputers. The rapid growth of Linux is partly due to it being a free clone of Unix, the most mature and powerful operating system of all. The other major factor in the speedy growth and adoption of Linux is the absence of barriers. Anyone can download and try it, and the source code is freely available to anyone who wants to use and contribute to it.

At the time it was a great example of form following function, like my first car. It ran, it was reliable, but it wasn't pretty, and it needed a lot of custom wiggling of this and jiggling of that to keep going. Running a Linux system back then meant learning your way around a hodgepodge of commands, scripts, and configuration files, and a fair bit of wiggling and jiggling. Software management, storage management, networking, audio, video, kernel management, process management…everything required a lot of hands-on work and continual study.

Some 17 years later, every important subsystem in Linux has substantially changed and improved. Now all those manual chores we had to do for basic administration are replaced by what I call "It Just Works Subsystems." Every aspect of running a Linux system is many times easier, and we can focus on using Linux to do cool things instead of having to wiggle and jiggle this and that just to keep it running.

I am delighted to present this greatly updated *Linux Cookbook* second edition, and I hope you enjoy learning about all of this cool new goodness.

Who Should Read This Book

This book is for people with some computer experience, though not necessarily Linux experience. I've done my best to make it as accessible as possible for Linux beginners. You should understand some basic networking concepts, such as IP addressing, Ethernet, WiFi, client, and server. You should know basic computer hardware and have some understanding of using the command line. If you need some help with these, there are abundant resources for learning them; I did not want to get bogged down in teaching material that is already well documented.

The recipes in this book are hands-on. My goal is for the reader to be successful on the first try, though don't feel badly if you are not. A general-purpose Linux computer is an extremely complex machine, and there is a lot to learn. Be patient, take your time, and read more than you want to. Chances are the answers you want are just a few sentences away.

Every Linux has built-in documentation for commands called *man pages* (short for "manual pages"). For example, *man 1 ls* documents the *ls*, or list directory contents, command. Type these commands exactly as shown in the book to open the correct man page. You can also find this information online.

Why I Wrote This Book

I have long wanted to write a book like this, that collects what I think are the most necessary Linux skills in one book. Linux is everywhere, and no matter where you find it, Linux is Linux, and the needed skills are the same. The tech world moves fast, and I think you will find this book provides a solid foundation that you can build on, no matter what direction your interests take you.

The cookbook format is especially good for teaching fundamentals because it shows how to solve specific real-world problems and separates the wordy explanations from the steps needed to accomplish a task.

Navigating This Book

This book is not a formal training course, where you start at the beginning and work your way to the end. Instead, you can jump in anywhere and hopefully find what you need.

It is roughly organized like this:

- Chapters 1, 2, and 3 cover installing Linux, managing the bootloader, stopping and starting, and answer the "Where do I get Linux and how do I make it go" questions.
- Chapter 4 provides an introduction to managing services with systemd, which is a big improvement from the old way of having to learn all manner of scripts, configuration files, and commands.
- Chapter 5 covers managing users and groups, Chapter 6 is about managing files and directories, and Chapter 7 covers backups and recovery. These three chapters are fundamental to system operations and security.
- Chapters 8, 9, and 11 are all about partitioning and filesystems, which are fundamental to managing data storage. Data management is the most important aspect of computing.
- Chapter 10 is fun. This chapter is about finding detailed information about your computer hardware without opening the case. Modern PC hardware self-reports a lot of information, and Linux supplements this self-reporting with databases of additional information.
- Chapters 12 and 13 teach setting up secure remote access, and Chapter 14 introduces the excellent firewalld, the dynamic firewall that easily handles all kinds of complicated scenarios, like roaming between different networks and managing multiple network interfaces.
- Chapter 15 introduces new features in CUPS, the Common Unix Printing System, including "driverless" printing, which is especially good for mobile devices because they can connect to a printer without having to download a lot of software.
- Chapter 16 shows how to control your own LAN name services with the excellent Dnsmasq. Dnsmasq has stayed current with support for new protocols, and the old commands and configuration options have not changed. It is a first-rate name server that seamlessly integrates DNS and DHCP for central management of IP addressing and advertising network services.
- Chapter 17 introduces chrony and timesyncd, two new implementations of the Network Time Protocol (NTP). It also includes the old tried-and-true *ntp* server and client.
- Chapter 18 introduces installing Linux on the Raspberry Pi, the popular little inexpensive single-board computer, and using it to build an internet firewall/gateway.

- Chapter 19 shows how to use SystemRescue to reset lost Linux and Windows passwords, rescue nonbooting systems, rescue data on a failing system, and customize SystemRescue to make it even more useful.
- Chapters 20 and 21 teach basic troubleshooting, with emphasis on searching log files, probing networks, and probing and monitoring hardware.
- The Appendix contains cheat sheets for managing software installation and maintenance.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions, as well as to refer to program elements such as Linux variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width`
: Used for program listings and some command options.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.



This element signifies a tip or suggestion.



This element signifies a general note.



This element indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, if example code is offered with this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but generally do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Linux Cookbook*, Second Edition, by Carla Schroder (O'Reilly). Copyright 2021 Carla Schroder, 978-1-492-08716-8."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

O'Reilly Online Learning



For more than 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, visit *http://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *https://oreil.ly/linux-cookbook-2e*.

Email *bookquestions@oreilly.com* to comment or ask technical questions about this book.

For news and information about our books and courses, visit *http://oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*.

Follow us on Twitter: *http://twitter.com/oreillymedia*.

Watch us on YouTube: *http://www.youtube.com/oreillymedia*.

Acknowledgments

I really lucked out with this book. My editor, Jeff Bleiel, has been unfailingly supportive and helpful, contributed numerous improvements, and kept the whole project organized and on track. If you think herding cats is difficult, try being a book editor.

My intern, Kate Urness, was a Linux newbie at the start of this adventure, which made her the perfect reviewer. She tested every recipe and contributed substantially to the accuracy and clarity of the recipes. We drank gallons of coffee and had fun together, which was also a substantial contribution.

Technical editor Daniel Barrett has a fabulous eye for detail, and relentlessly pressed for more precision in wording and descriptions, and provided many improvements. Providing a book full of commands is the easy part, explaining how they work is the hard part. Every writer should be fortunate to have such a technical editor.

Technical editor Jonathan Johnson found things everyone else missed, contributed some super-cool command incantations, and provided humor, and let me tell you, I needed it.

Zan McQuade, acquisitions editor, started this whole crazy deal. There were talks over the years about updating the *Linux Cookbook*, but Zan made it happen.

Great thundering herds of thanks to my wife, Terry, who fed the mules and cats and dogs and me, provided much encouragement, and kept me from running away from home because I lost my mind and agreed to write another book.

Special thanks to our cats, Duchess (Figure P-1), Stash (Figure P-2), and Mad Max (Figure P-3), who appear in this book. They helped greatly by sleeping on my keyboard, not letting me have my chair, and making mysterious loud crashes a lot.



Figure P-1. Duchess holds my feet down, for my own good



Figure P-2. Stash cat, our glamour boy

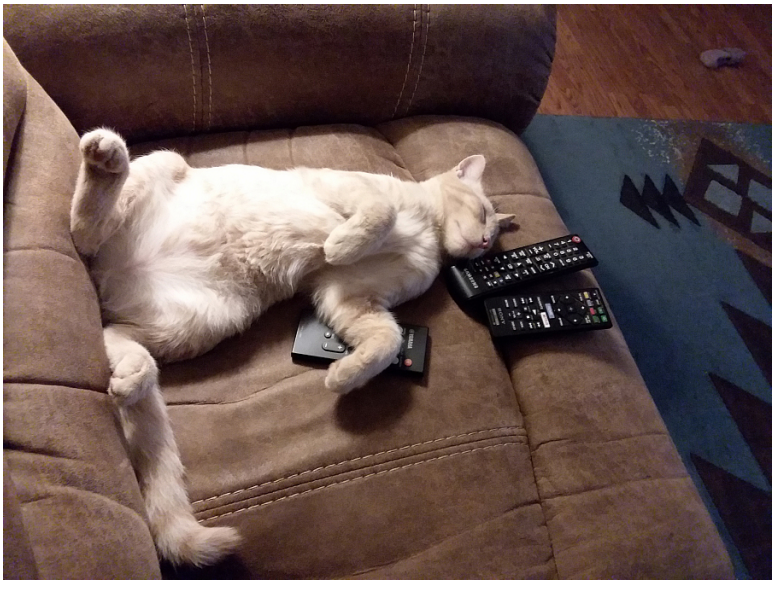


Figure P-3. Mad Max rests up for the next mayhem

CHAPTER 1
Installing Linux

One of the hurdles for new Linux users is installing Linux. Linux is the easiest computer operating system to install: pop in your installation disk, answer a few questions, and then do something else until it finishes. In this chapter you will learn how to install Linux by itself, how to run a live Linux, how to multiboot multiple Linux distributions on one computer, and how to dual-boot with Microsoft Windows.



Experimenting with Linux

You need the freedom to make mistakes, so if it is possible, use a second computer for getting acquainted with Linux. If this is not possible, make sure you always have fresh backups of your data. You can always restore a broken Linux installation, but your data is irreplaceable. If you are setting up dual-boot with Windows, be sure you have your Windows installation and recovery media.

Most Linux distributions provide dual-purpose installation images: you can run them live from a USB stick and install them to your hard drive from the same image. A live Linux makes no changes to your computer—just boot it up, check it out, then reboot to your host system. Some live Linuxes, such as Ubuntu, support storing your data on the USB stick, so you have a completely portable Linux that you can run from any computer.

Multiboot is installing more than one operating system on a computer, and then picking the one you want to use from your boot menu. You can multiboot any Linux system, any of the free Unixes (FreeBSD, NetBSD, OpenBSD), and you can multiboot Linux and Microsoft Windows. Dual-booting Linux and Windows is a common way for Windows users to get acquainted with Linux, and for users who need both.

What about Apple's macOS, you ask? Sorry, but dual-booting Linux and macOS is an unreliable endeavor that gets more difficult with every macOS release. One alternative for running both on a single machine is to run Linux in Parallels, the macOS virtual machine host.

Rather than installing Linux yourself, you could buy a PC with Linux already installed. There are a number of good Linux specialists that sell Linux laptops, desktops, and servers. System76, ZaReason, Linux Certified, Think Penguin, Entroware, and Tuxedo Computers are all Linux specialists. Dell has been expanding its Linux lineup, and enterprise Linux vendors Red Hat, SUSE, and Ubuntu all partner with hardware vendors, including Dell, Hewlett-Packard, and IBM.

Still, it pays to know how to install Linux. This opens up a whole world of experimentation, customization, and disaster recovery. *Distro-hopping* is a time-honored pastime, where you download and try different Linux distributions.

Even though installing Linux takes just a few steps, you need a certain amount of knowledge, especially when you want to customize your installation, like setting up disk partitions in a particular way or multibooting with other Linux distributions or with Microsoft Windows. You need to know how to enter your system Basic Input Output System (BIOS) or Unified Extensible Firmware Interface (UEFI) setup. You need good internet access. All Linux distributions are freely downloadable, even the commercial enterprise distributions like Red Hat, SUSE, and Ubuntu. Download sizes range from a few megabytes for super-small Linuxes like Tiny Core Linux, which bundles a complete operating system with a graphical desktop into 12 MB, to 10+ GB for SUSE Linux Enterprise Server. Most Linux distros provide 2–4 GB installation images, which fit perfectly on a DVD or small USB stick.

Most Linux distributions provide a network installer image; for example, Debian's is around 200 megabytes. This installs enough of a Debian system to boot up, connect to the internet, and then download only the packages you want, rather than downloading a complete installation image.

You can freely share any Linux distribution that you download.

You also have the option to buy Linux distributions on DVD and USB media. Visit Shop Linux Online and Linux Disc Online to find physical installation media for various Linux distributions.

Booting from Installation Media

You must be able to boot your system from your USB or DVD installation disk. You may have to enter your system's BIOS or UEFI setup to enable booting from a removable device. Some have an option to select an alternate boot device without entering the BIOS/UEFI; for example, my laptop's UEFI displays a screen at startup that lists all

the relevant keypresses: F2 or Delete to enter setup, and F11 to enter the alternate boot device menu. Dell systems use F12 to open their one-time boot menu. Every one is special and unique, so check out your motherboard manual to learn how.

You may have to disable Secure Boot in your UEFI setup to enable booting from removable media. Fedora, openSUSE, and Ubuntu all have their own signing keys and will boot with Secure Boot enabled. Other Linux distributions, such as SystemRescue (Chapter 19), do not.



Secure Boot

Secure Boot is a security feature of your UEFI setup. When Secure Boot is enabled, it allows booting only operating systems that provide special signed keys. The idea is to prevent malicious code from controlling your bootloader.

Most Linux distributions do not provide signed keys, so Secure Boot must be disabled to run them.

Where to Download Linux

There are hundreds of Linux distributions, and a great place to learn about them is DistroWatch.com, the most comprehensive Linux distribution resource. DistroWatch publishes reviews, detailed information, and news, and their popular list of the top 100 distributions.

Best Linux for Newbies

Linux provides rather a lot of a good thing, maybe too much. The recipes in this book were tested on openSUSE, Fedora Linux, and Ubuntu Linux. These three are well-established, popular, and well-maintained, and they represent three different Linux families (see the Appendix). In my experience Ubuntu is perfect for a Linux newbie because it has the easiest installer, good documentation, and a large and supportive user community.

Every Linux has its differences: different software installers, different defaults, different file locations…but the fundamentals are all similar. Most of what you learn on any particular distro is applicable to all of them.



Hardware Architectures

How-to authors used to be able to take it for granted that readers were using x86 hardware. With the rise in popularity of ARM processors this is no longer true. Linux supports a large number of hardware architectures, and Recipe 10.11 shows how to detect what you have. You cannot accidentally install the wrong Linux version because the installation will fail at the beginning, and you will see an error message telling why.

Linux installation images are packaged in the ISO 9660 format, and have a **.iso* extension, for example, *ubuntu-20.04.1-desktop-amd64.iso* for x86-64 machines, and *ubuntu-20.04.1-live-server-arm64.iso* for ARM machines. This is a compressed archive that contains the entire filesystem and the installation program. When you copy this archive to your installation medium, it is uncompressed and you can see all the files.

The **.iso* format was originally for CDs and DVDs. Once upon a time, Linux fit on a single CD. (Once upon a time, it fit on a few 3.5" diskettes!) Now most Linux distributions are too large for CDs. USB sticks are perfect for Linux installations, as they are inexpensive, reusable, and much faster than optical media.

1.1 Entering your System BIOS/UEFI Setup

Problem

You want to enter your system's BIOS/UEFI setup.

Solution

Enter your BIO/UEFI setup by pressing the appropriate F*n* key at startup. On Dell, ASUS, and Acer systems this is usually F2, and Lenovo uses F1. However, this varies; for example, some systems use the Delete key, so check your machine's documentation. Some systems tell which key to press on their startup screens. It can be a bit tricky to press the key at the right time, so start pressing it right after you press the power button, just like banging on an elevator button to make it arrive faster.

Every UEFI looks different; for example, Lenovo's is bright and well organized (Figure 1-1).

The ASRock UEFI on my test system is dark and dramatic (Figure 1-2). This particular motherboard targets gamers and has many settings for overclocking the CPU and other performance enhancements. This screen shows the motherboard browser; hover the cursor over any item and it provides information about that item.

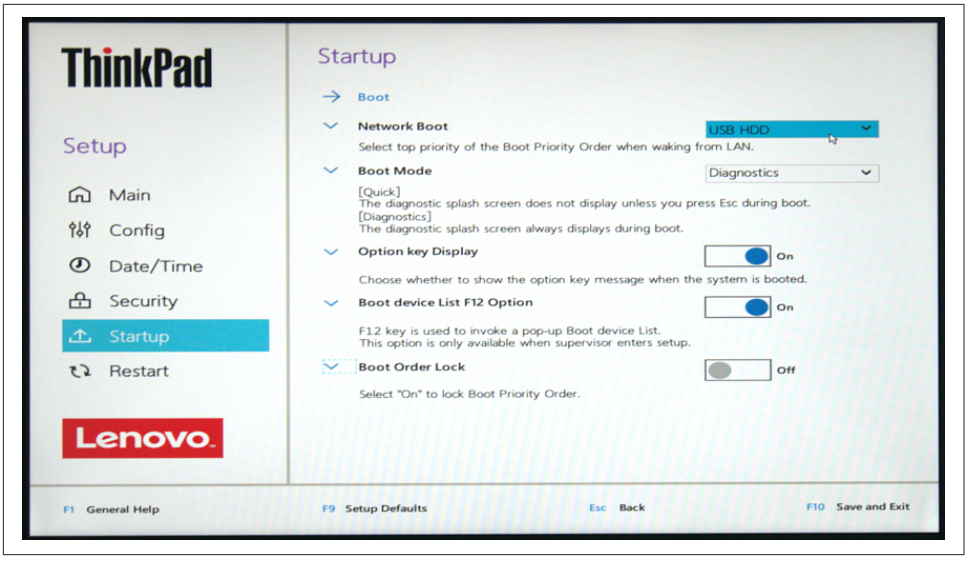


Figure 1-1. Lenovo's UEFI on a new ThinkPad

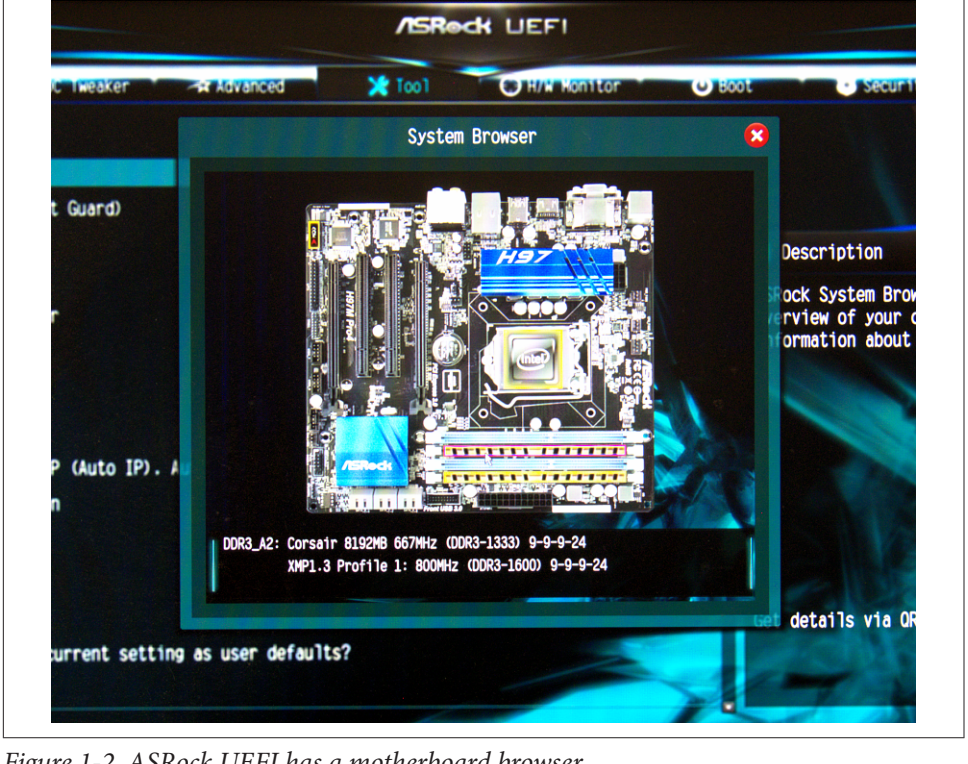


Figure 1-2. ASRock UEFI has a motherboard browser

Discussion

When you boot up your computer, the first startup instructions come from the BIOS or UEFI firmware, stored on the computer's motherboard. BIOS is the old legacy system that has been with us since 1980. UEFI is its modern replacement. UEFI includes legacy BIOS support. Nearly all computers made after the mid-2000s have UEFI.

UEFI has considerably more features than the old BIOS and is like a little operating system. The UEFI setup screens control boot order, boot devices, security options, Secure Boot, overclocking, displaying hardware health, networking, and many more functions.

See Also

- The documentation for your motherboard
- Unified Extensible Firmware Interface Forum

1.2 Downloading a Linux Installation Image

Problem

You want to find and download a Linux installation image.

Solution

First, you need to choose which Linux you want to try. If you don't know where to start, I recommend Ubuntu Linux. Fedora Linux and openSUSE Linux are also excellent for anyone new to Linux.

After your download is completed, verify that you have a good download. This is an important step that ensures you have an image that did not get damaged during the download or was otherwise altered in some way.

Every Linux distribution provides signed keys and checksums for its download images. Ubuntu provides copy-and-paste instructions. Open a terminal and change to the directory where you downloaded Ubuntu. For Ubuntu 21.04, verifying the installation image looks like this:

```
$ echo "fa95fb748b34d470a7cfa5e3c1c8fa1163e2dc340cd5a60f7ece9dc963ecdf88 \
*ubuntu-21.04-desktop-amd64.iso" | shasum -a 256 --check

ubuntu-21.04-desktop-amd64.iso: OK
```

If you see "shasum: WARNING: 1 computed checksum did NOT match," then your download is no good, assuming you copied the correct checksum. In most cases the image was corrupted during the download, so try downloading it again.

Other Linux distributions provide slightly different verification methods, so follow their instructions.

Discussion

A great site to learn about the hundreds of Linux distributions is Distrowatch.com. Distrowatch provides news and information about more Linux distros than anyone.

See Also

- *man 1 sha256sum*
- Ubuntu Linux
- Fedora Linux
- openSUSE Linux

1.3 Creating a Linux Installation USB Stick with UNetbootin

Problem

You have downloaded a Linux installation **.iso* image, and you want to transfer it to a USB stick to create your own installation media. You prefer a graphical tool to create your installation media.

Solution

Try UNetbootin, Universal Netboot Installer. It runs on Linux, macOS, and Windows, so you can download and create a Linux installation disk on any of these operating systems. UNetbootin creates an installation USB drive from your downloaded **.iso* file or downloads a fresh **.iso* file (Figure 1-3).

You may use any size USB stick (that is larger than your **.iso* file, of course). The **.iso* overwrites the entire device, so you can't use it for anything else, and you must use a separate USB stick for each **.iso* file.

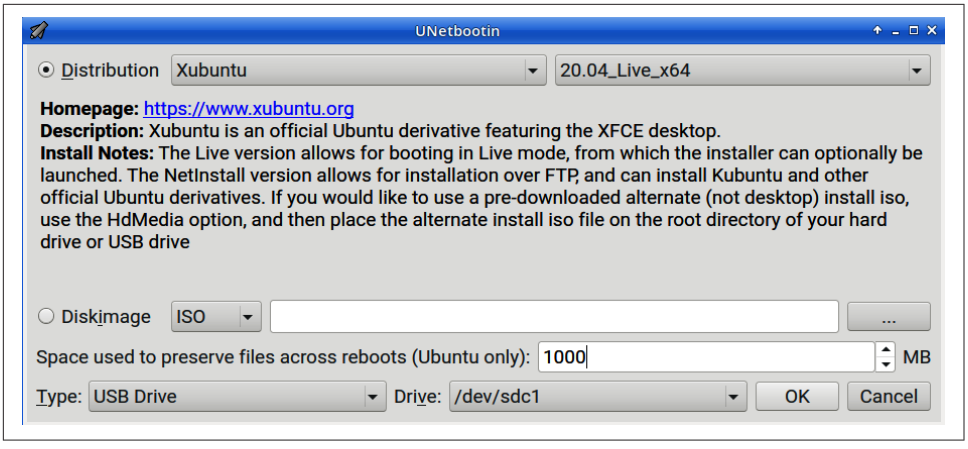


Figure 1-3. Using UNetbootin to create an installable Linux USB stick

The UNetbootin site provides downloads and instructions. Some Linux distributions provide UNetbootin packages, but downloading from the UNetbootin site is simple and always up-to-date.

Discussion

Other good graphical apps are USB Creator, ISO Image Writer, and GNOME Multi-Writer, which copies to multiple USB drives at one time.

After you have created your installation USB drive, you can view the files on it. The single **.iso* expands into a complete filesystem full of files and directories, like this example for Ubuntu:

```
$  ls -C1 /media/duchess/'Ubuntu 21.04.1 amd64'/
boot
casper
dists
EFI
install
isolinux
md5sum.txt
pics
pool
preseed
README.diskdefines
ubuntu
```

Every Linux distribution sets up its installers in its own way. This example shows Fedora's installation files:

```
$ ls -C1 /media/duchess/Fedora-WS-Live-34-1-6/
EFI
images
isolinux
LiveOS
```

It would be lovely to have a single USB stick with a herd of Linux installation files, and there are some programs that do this. My favorite is Ventoy. Ventoy supports a large number of Linux distributions. It runs on both Linux and Windows, and you can create a USB stick full of Linux installers for running live Linuxes, and for permanent installations to hard disks.

See Also

- UNetbootin
- Chapter 9
- Ventoy

1.4 Creating a Linux Installation DVD with K3b

Problem

You want to create a Linux DVD installation disk with a graphical tool.

Solution

Use K3b (KDE Burn Baby Burn). K3b is the best graphical CD/DVD writing app for Linux.

If you do not have a Linux system, then use any CD/DVD writing program that writes an ISO 9660 image. Your chosen CD/DVD writer will phrase this as something like "burn an existing image to disk."

You will see something like Figure 1-4 on K3b. Click "Burn Image," and note the confirmation on the bottom left that says "Write an ISO 9660…image to an optical disk."

On the next screen (Figure 1-5), in the top left drop-down selector, select your **.iso* image. Then on the top right select "ISO 9660 filesystem image." At the bottom under Settings, check "Verify written data." This computes a checksum after your image is written and compares it to the checksum of the original **.iso*. This is an important step because if the checksums do not match you have a corrupted disk, which is unusable.

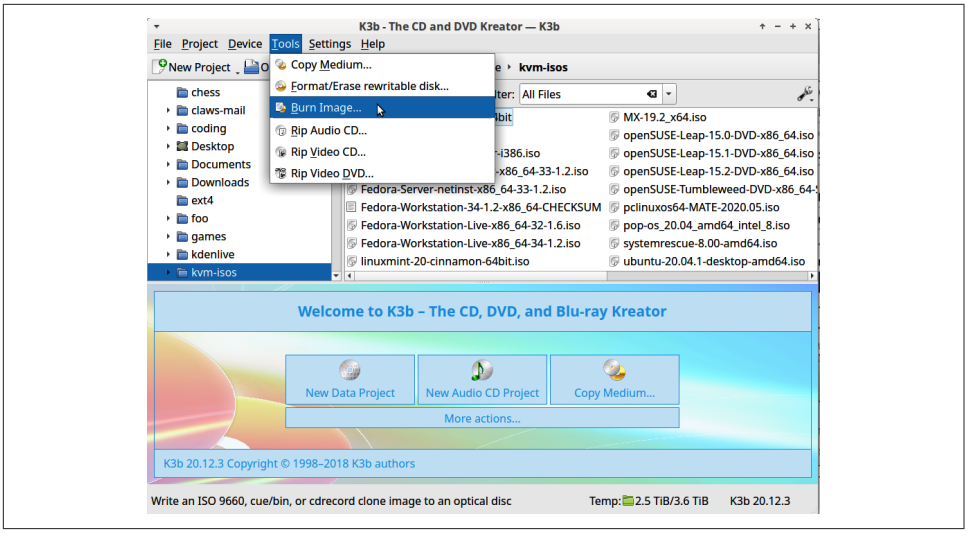


Figure 1-4. Creating an installation DVD with K3b

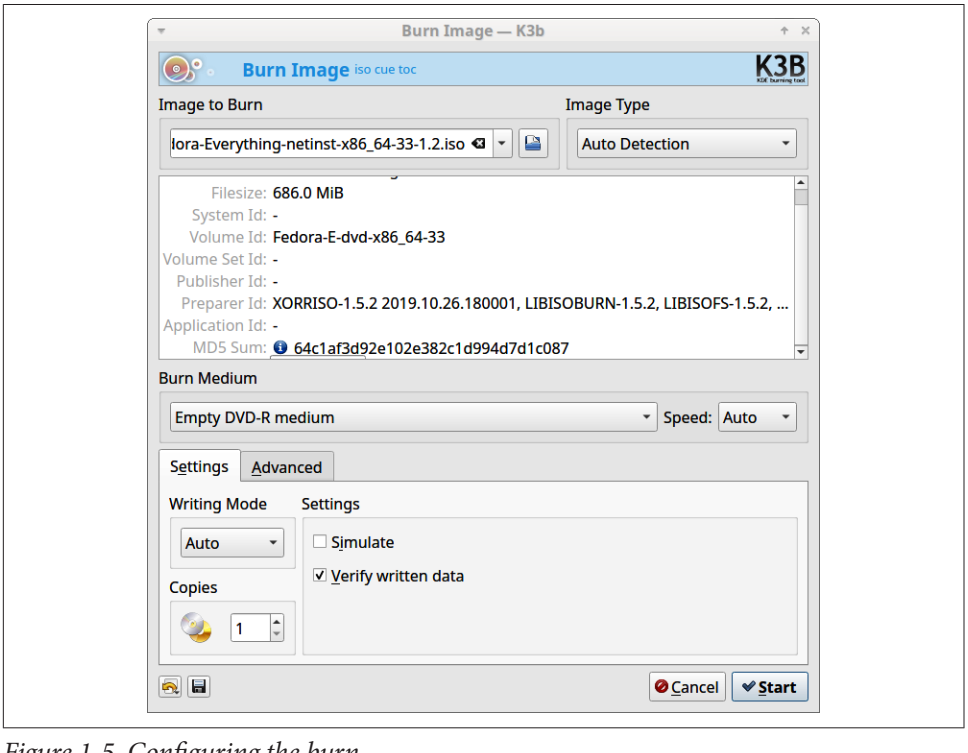


Figure 1-5. Configuring the burn

When the disk is successfully written, you will see a success message like the one shown in Figure 1-6. If there were any errors, this screen will show some helpful error messages.

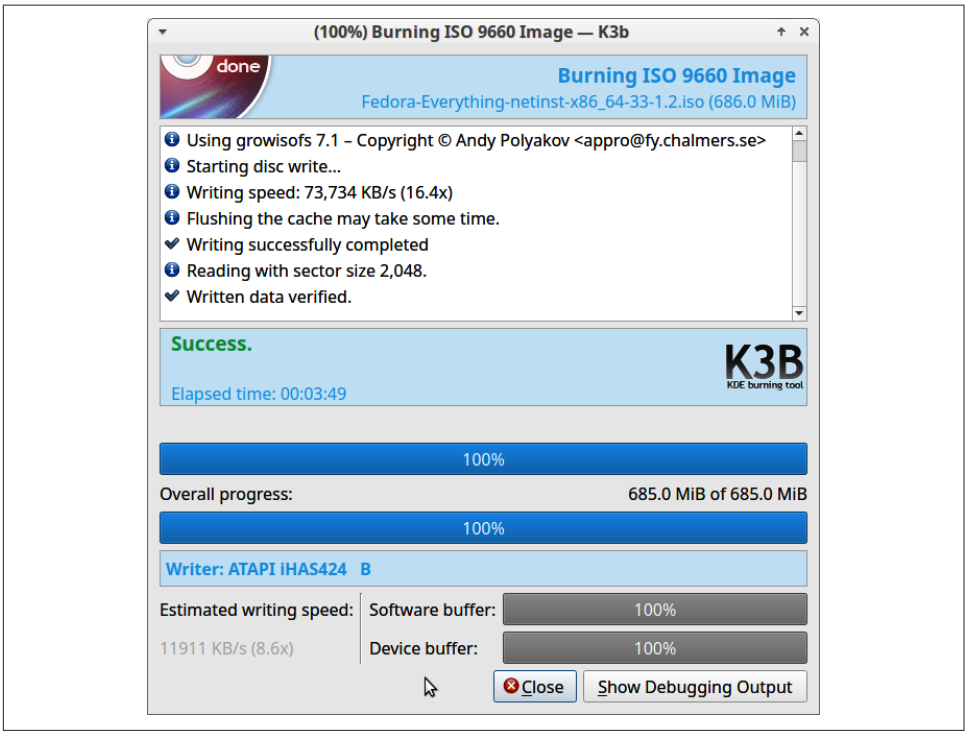


Figure 1-6. Success!

Discussion

Brasero and XFBurn are also great CD/DVD writing apps for Linux, with simpler interfaces than K3b, but still plenty of functionality.

The tech world changes fast. Just a few years ago I was burning CDs and DVDs for everything. Then USB devices swept the market, and I haven't burned a disk for years, until I started writing this chapter.

Take heart, for CDs and DVDs are not obsolete, despite the efforts of computer makers to make them obsolete by not including CD/DVD drives in their machines. This is not a problem because you can purchase an external USB CD/DVD drive. You can even find bus-powered drives, so you only need a USB cable, and don't have to hassle with a power cable. CD/DVD blanks are still good quality, so if you prefer optical disks, they are a reliable choice.

See Also

- K3b
- Brasero

1.5 Using the wodim Command to Create a Bootable CD/DVD

Problem

You want a command-line tool to create a bootable CD/DVD.

Solution

Try the *wodim* command. Your optical drive is most likely */dev/cdrom*, symlinked to */dev/sr0*. Use the symlink because it has correct permissions:

```
$ ls -l /dev | grep cdr
lrwxrwxrwx  1 root root           3 Mar  7 12:38 cdrom -> sr0
lrwxrwxrwx  1 root root           3 Mar  7 12:38 cdrw -> sr0
crw-rw----+ 1 root cdrom    21,   2 Mar  7 08:34 sg2
brw-rw----+ 1 root cdrom    11,   0 Mar  7 12:57 sr0
```

Then copy your installation image to disk:

```
$ wodim dev=/dev/cdrom -v ubuntu-21.04-desktop-amd64.iso
```

Discussion

In the *ls -l* example, there are *sg2* and *sr0* devices. *sg2* is a character device, and *sr0* is a block device. Character devices provide raw access to a hardware device using the raw kernel drivers. Block devices provide buffered access to a hardware device through various software programs that handle reading and writing to physical media. Users interact with storage devices, like DVDs and hard disks, through the kernel's block device drivers. You can see raw and block kernel modules listed in your */boot/config-** file.

See Also

- *man 1 wodim*

1.6 Creating a Linux Installation USB Stick with the dd Command

Problem

You want to create your installation USB drive from the command line, rather than using a graphical tool.

Solution

Use the *dd* command. *dd* is on every Linux and works the same way on all of them.

First, verify the dev name for your USB stick with the *lsblk* command, so that you copy your image to the correct device. In the following example, that is */dev/sdb*:

```
$ lsblk -o NAME,FSTYPE,LABEL,MOUNTPOINT

NAME   FSTYPE  LABEL       MOUNTPOINT
sda
├─sda1 vfat                /boot/efi
├─sda2 xfs     osuse15-2   /boot
├─sda3 xfs                 /
├─sda4 xfs                 /home
└─sda5 swap                [SWAP]
sdb
└─sdb1 xfs     32gbusb
sr0
```

The following example creates a USB installation stick and shows its progress:

```
$ sudo dd status=progress if=ubuntu-20.04.1-LTS-desktop-amd64.iso of=/dev/sdb
211509760 bytes (212 MB, 202 MiB) copied, 63 s, 3.4 MB/s
```

This takes a few minutes. When it's finished, it looks like this:

```
2782257664 bytes (2.8 GB, 2.6 GiB) copied, 484 s, 5.7 MB/s
5439488+0 records in
5439488+0 records out
2785017856 bytes (2.8 GB, 2.6 GiB) copied, 484.144 s, 5.8 MB/s
```

Remove the drive, then reinsert it and take a quick look at the files. Figure 1-7 shows the installation files for Ubuntu Linux in the Thunar file manager.

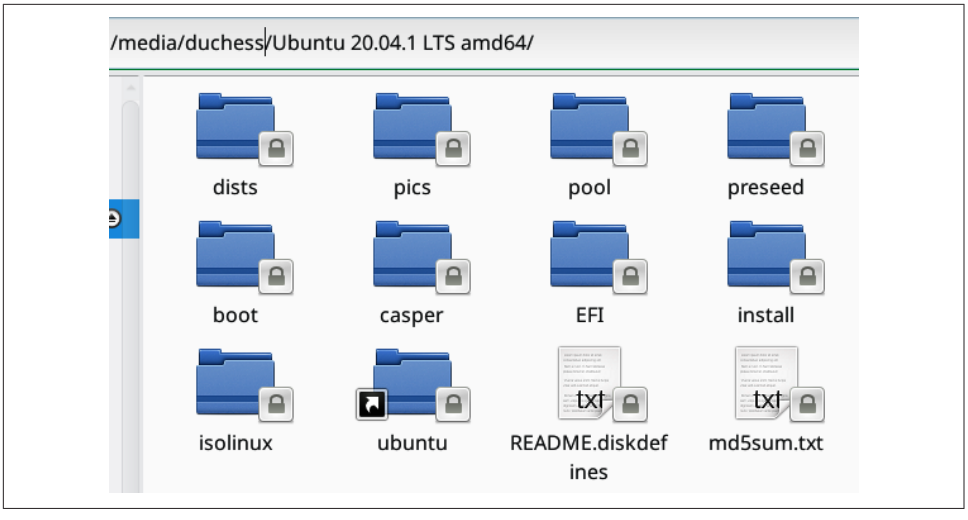


Figure 1-7. Ubuntu Linux installation files in Thunar

The files are marked with padlocks because the Ubuntu installer uses the SquashFS read-only filesystem. You can read them, but not delete or edit them.

The installation USB stick is ready to use.

Discussion

Identifying the correct device to copy your installation to is super important. In the *lsblk* example, there are only two storage devices. Note the LABEL column; you can create labels on filesystems so you know what they are. (See Recipe 9.2 and the filesystem creation recipes in Chapter 11 to learn how to create filesystem labels.)

The graphical installation media creators are good, but I prefer the *dd* command because it is simple and reliable. *dd* is short for Disk Duplicator. This is one of those ancient useful GNU commands, in the GNU *coreutils* package, that has been around forever.

See Also

- *man 1 dd*

1.7 Trying a Simple Ubuntu Installation

Problem

You want to try a simple Ubuntu installation. You have your installation medium ready, and you know how to boot to your installation medium. There is nothing on the computer that you want to keep, so Ubuntu can take over the hard drive.

Solution

The following example demonstrates a fast, simple installation of Ubuntu Linux 21.04, Hirsute Hippo. All Ubuntu releases have alliterative animal names.

Insert your installation device, power on the machine, and open your system's one-time boot menu. Select the installation device and boot up (Figure 1-8).

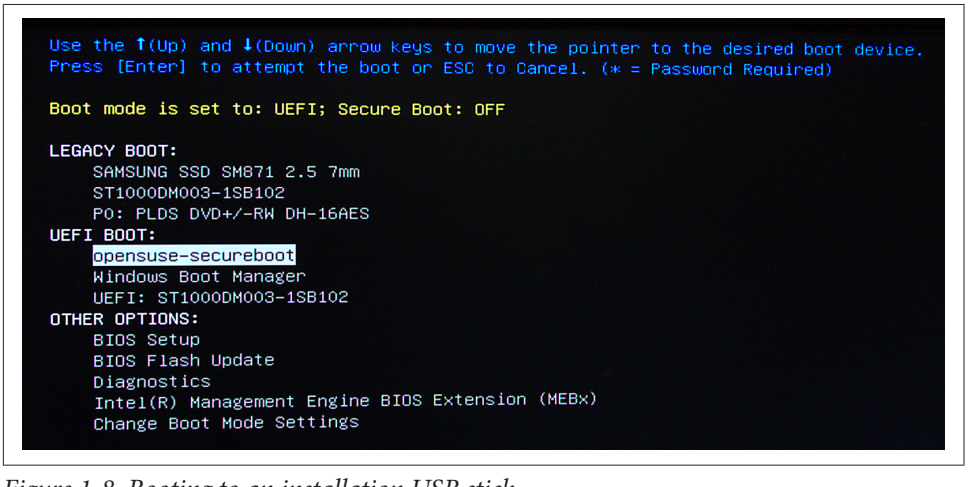


Figure 1-8. Booting to an installation USB stick



All UEFI Screens Look Different

Every vendor's UEFI looks different, and every release changes in appearance. The preceding example is a Dell UEFI one-time boot screen.

When the GRUB menu appears, select the default option. For Ubuntu 21.04 this is *Ubuntu*, and it boots by default when you make no selection (Figure 1-9).

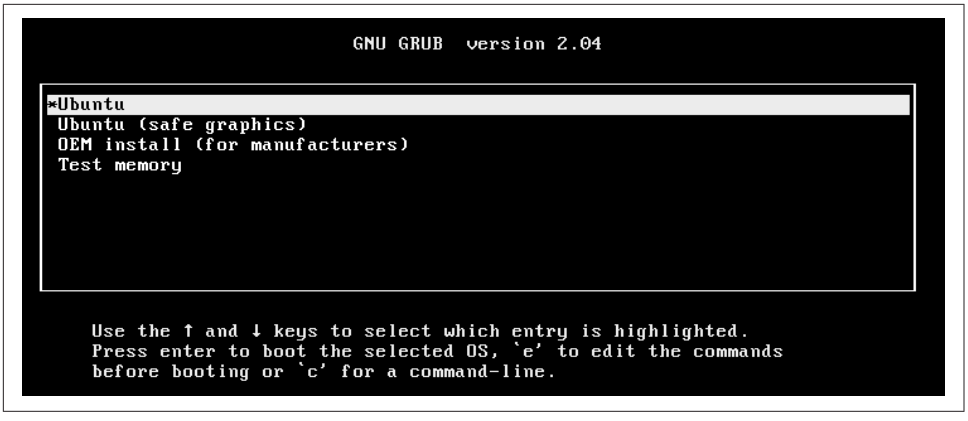


Figure 1-9. Ubuntu installer's GRUB boot menu

Then you will have a choice of Try Ubuntu and Install Ubuntu. Try Ubuntu launches the live version, and Install Ubuntu opens the installer (Figure 1-10). It doesn't matter which one you select because there is a big installation button on the live desktop.

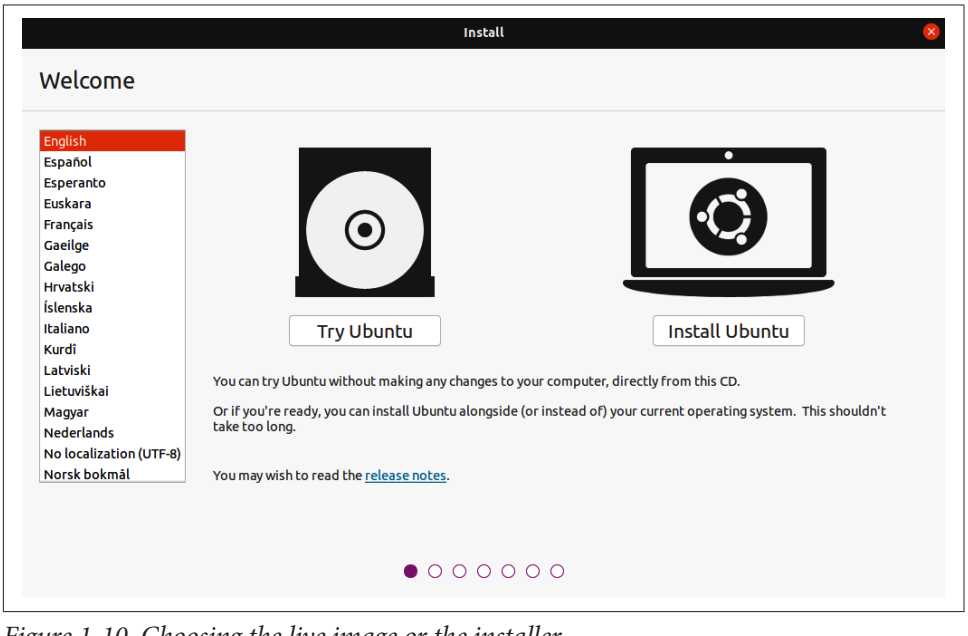


Figure 1-10. Choosing the live image or the installer

When you launch the installer, it walks you through a few steps. First, language and keyboard layout.

Then, if you have a wireless network interface, you have the option to set it up or wait until after installation.

Then configure your installation. On the "Updates and other software" screen, select "Normal installation" (Figure 1-11).

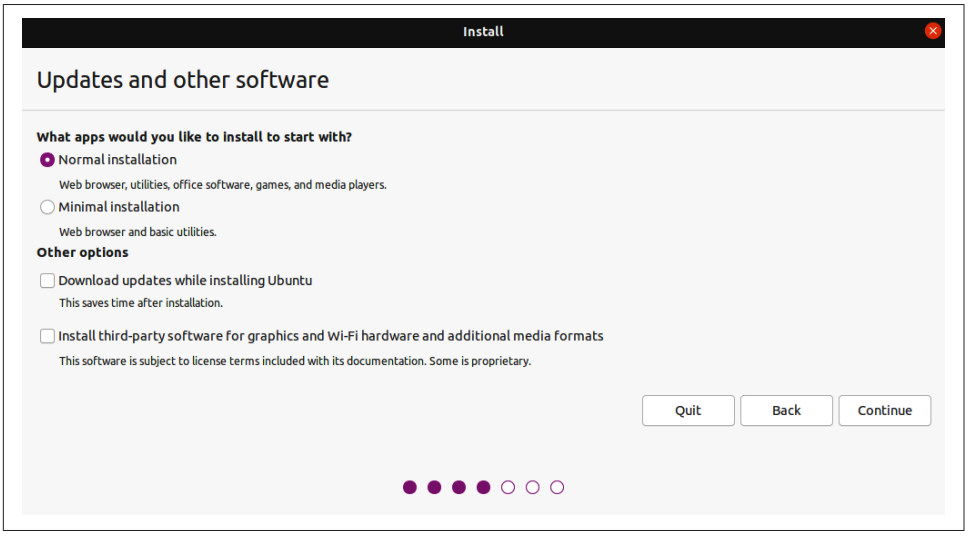


Figure 1-11. Select Normal installation

On the next screen, select "Erase disk and install Ubuntu," then click Install Now (Figure 1-12).

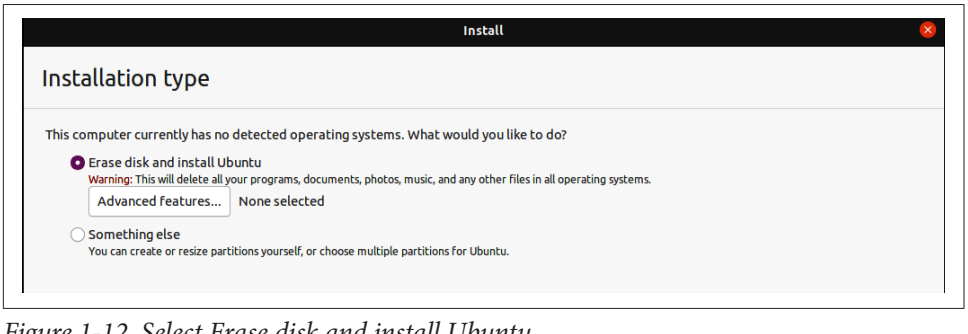


Figure 1-12. Select Erase disk and install Ubuntu

The next screen asks "Write the changes to disk?" Click Continue. There are a few more screens: set your time zone, create your username, password, and hostname, and then the installation starts. You do not have to do anything until the installation is finished, then restart the computer, remove your installation device when prompted, and press the Enter key. After restart you will have a few setup screens, then you can play with your nice new Ubuntu Linux.

Discussion

Most Linux distributions have a similar installation process: boot your installation medium, then choose a default simple installation or choose a customizable installation. Some ask all the questions at the beginning, such as username and password; others do the final setup after reboot.

Linux installers typically have back buttons to go back and make changes. You may quit at any time, though this may leave your system in an unusable state. This is not fatal; just start over and do a complete installation.

You may reinstall as many times on as many machines as you want without worrying about license keys, except for the enterprise distros that require registration keys (Red Hat, SUSE, or Ubuntu with paid support).

See Also

- Ubuntu documentation

1.8 Customizing Partitioning

Problem

You want to set up your own partitioning scheme.

Solution

In this recipe we will redo the example Ubuntu installation in Recipe 1.7 and set up our own partitioning scheme.



Entire Disk Will Be Erased

In this recipe a new partition table is created, which erases the entire disk.

You can set up partitioning in any number of ways. Table 1-1 shows how I prefer to set up my Linux workstations.

Table 1-1. Example partitioning scheme

Partition name	Filesystem type	Mountpoint
/dev/sda1	ext4	/boot
/dev/sda2	ext4	/

Partition name	Filesystem type	Mountpoint
/dev/sda3	ext4	/home
/dev/sda4	ext4	/tmp
/dev/sda5	ext4	/var
/dev/sda6	swap	

When you get to the Installation Type screen, select "Something else" to start your customized installation (Figure 1-13).

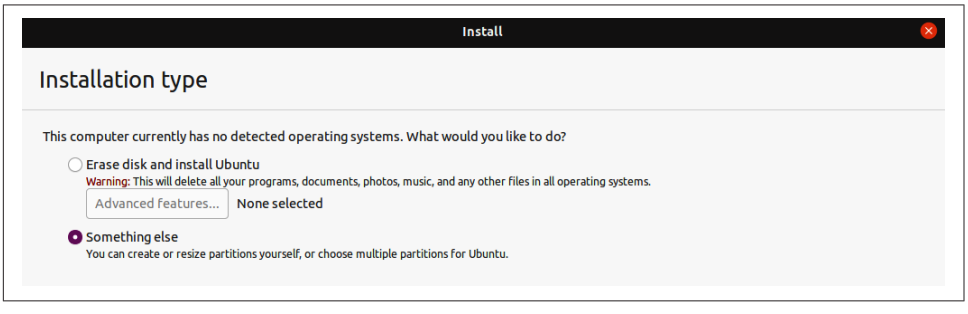


Figure 1-13. Selecting the installation type

When you get to the partitioning screen, erase the entire disk by clicking New Partition Table. Then you will see something like Figure 1-14.

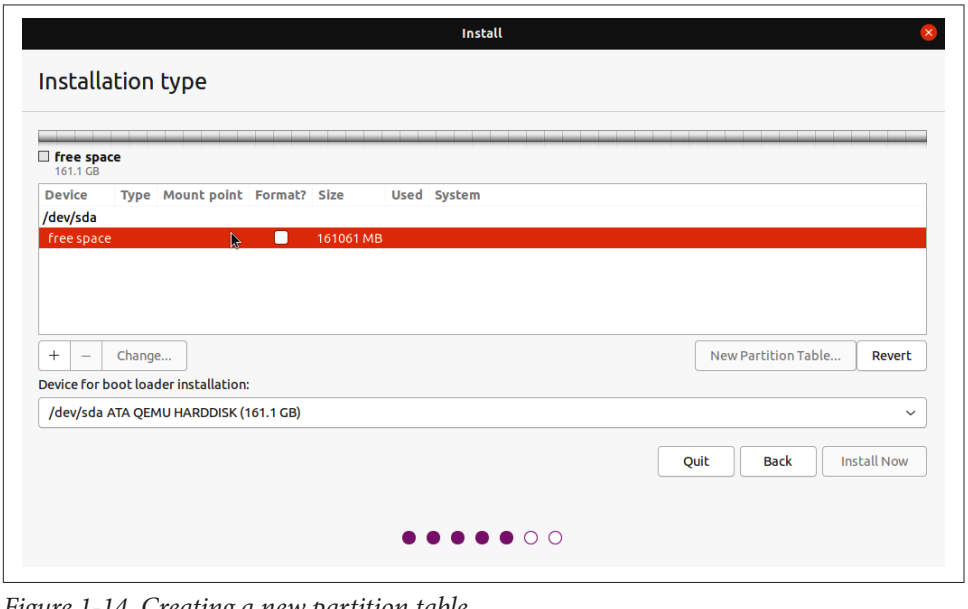


Figure 1-14. Creating a new partition table

To create new partitions click on the “free space” line to select it, then click the plus sign, +, to add a new partition. Set the size, filesystem, and mountpoint. Figure 1-15 creates a 500 MB */boot* partition.

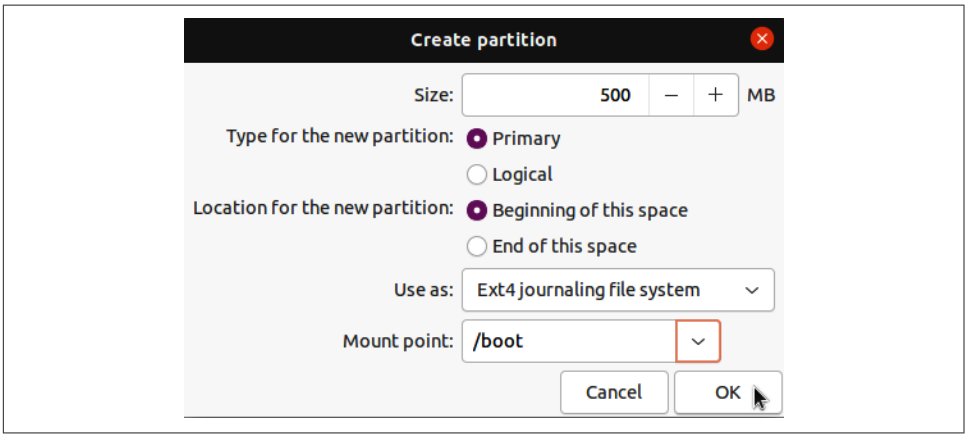


Figure 1-15. Creating the boot partition

Click on “free space” again, click the plus sign, and keep going until you have created all of your partitions. Figure 1-16 shows the result: */boot*, */home*, */var*, */tmp*, and the swap file are all on their own partitions.

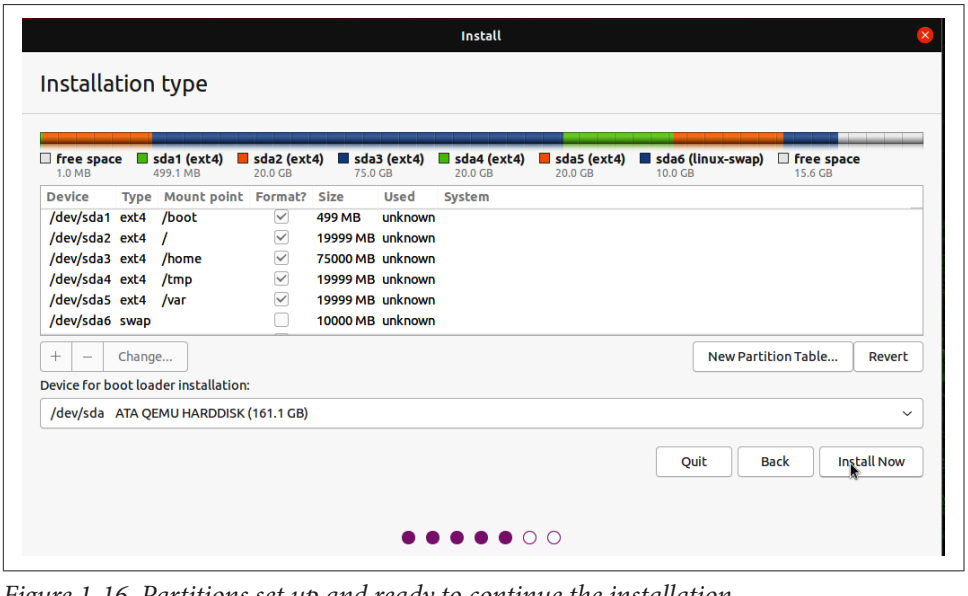


Figure 1-16. Partitions set up and ready to continue the installation



Selecting Partitions to Format

Note the "Format?" checkboxes in the partitioning screen. All new partitions must be formatted with a filesystem.

Discussion

The examples are from a virtual machine, so the hard disk is *vda* instead of *sda*.

The example in the recipe uses the Ext4 filesystem on all the partitions. You may use whatever filesystems you want; see Chapter 11 to learn more.

Disk partitions are like having a bunch of separate physical disks. Each one is an independent section of the hard disk, and each partition can have a different filesystem. The filesystems you select, and their sizes, all depend on how you use your system. If you need a lot of data storage, then */home* needs to be large. It could even be a separate disk.

Giving */boot* its own partition makes managing multiboot systems easier because this makes the boot files independent of whatever operating systems you install or remove. 500 MB is more than enough.

Putting /, root, in its own partition makes it easy to restore or to nuke and replace with a different Linux. 30 GB is more than enough for most distros, except when you use the Btrfs filesystem, then you should make it 60 GB to make room for storing snapshots.

Put */home* on its own partition to isolate it from the root filesystem, so you can replace your Linux installation without touching */home*. */home* could even be on a separate disk.

/var and */tmp* can fill up from runaway processes. Putting them on their own partitions prevents them from crashing the other filesystems. Mine are usually 20 GB each, and for a busy server they need to be larger.

Putting a swap file equal to the size of your RAM on its own partition enables suspend-to-disk.

See Also

- The discussion in Recipe 3.9 to learn about suspend and sleep states
- Chapter 8
- Chapter 9

1.9 Preserving Existing Partitions

Problem

You have */home* on its own partition and want to preserve it for the new Linux installation.

Solution

In Recipes 1.7 and 1.8 we erased the existing installation by creating a new partition table. When you have partitions that you want to preserve, such as */home* or any shared directory, do not create a new partition table. Instead, edit the existing partitions. You can delete existing partitions, create new partitions, and reuse existing partitions.

In the following example in the Ubuntu installer, */dev/sda3* is a separate */home* partition. Right-click on it, then click "Change…" Then you can set its mountpoint to */home*, and make sure that the "Format?" box is NOT checked (Figure 1-17). If you format it or change the filesystem type, all data on the partition will be erased.

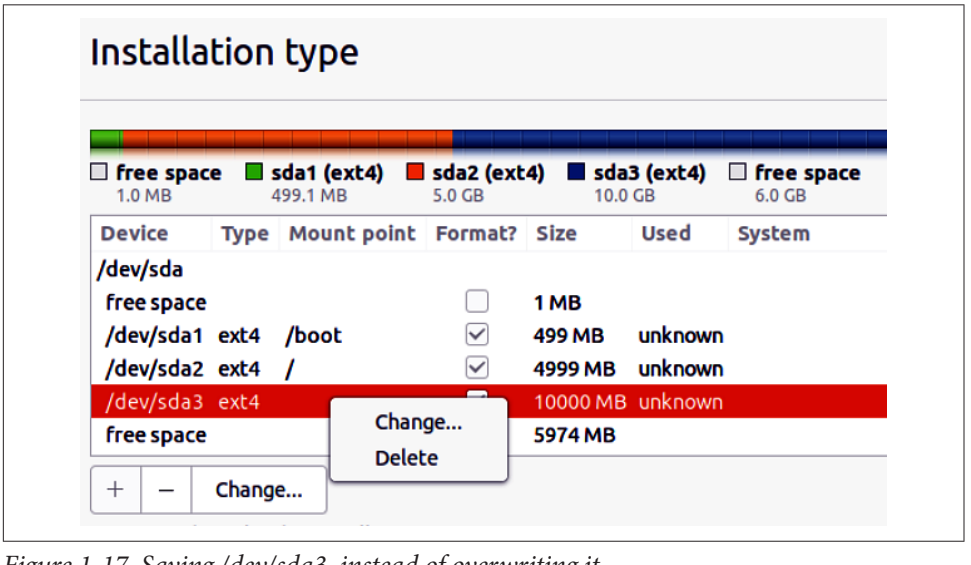


Figure 1-17. Saving /dev/sda3, instead of overwriting it

Discussion

The Discussion in Recipe 1.8 goes into some detail on customizing your partitioning scheme and which filesystems benefit from being isolated on their own partitions.

See Also

- Chapter 8
- Chapter 9

1.10 Customizing Package Selection

Problem

You don't want the default package installation, but prefer to select your own software to install. For example, you might want to set up a development workstation, a web server, a central backup server, a multimedia production workstation, a desktop publishing workstation, or choose your own office productivity applications.

Solution

Every Linux distro manages installation options a little differently. In this recipe you will see examples for openSUSE and Fedora Linux. openSUSE supports multiple installation types from a single installation image, and Fedora Linux has several different installation images.

These two examples are typical of the general-purpose Linux distributions.

Remember, you can install and remove software all you want after installation.

openSUSE

The openSUSE installer supports both a simple installation from defaults and extensive customization options. It has two screens that control package selection. The first screen (Figure 1-18) provides a selection of system roles to choose from, such as a desktop system with the KDE or GNOME graphical environment, a generic desktop with the IceWM window manager, a server with no graphical environment, or a transactional server with no graphical environment. Each role comes with a prefab set of packages. You can install one of these as is or select packages to install or remove.

Each role is customizable, as you will see a few screens later (Figure 1-19).

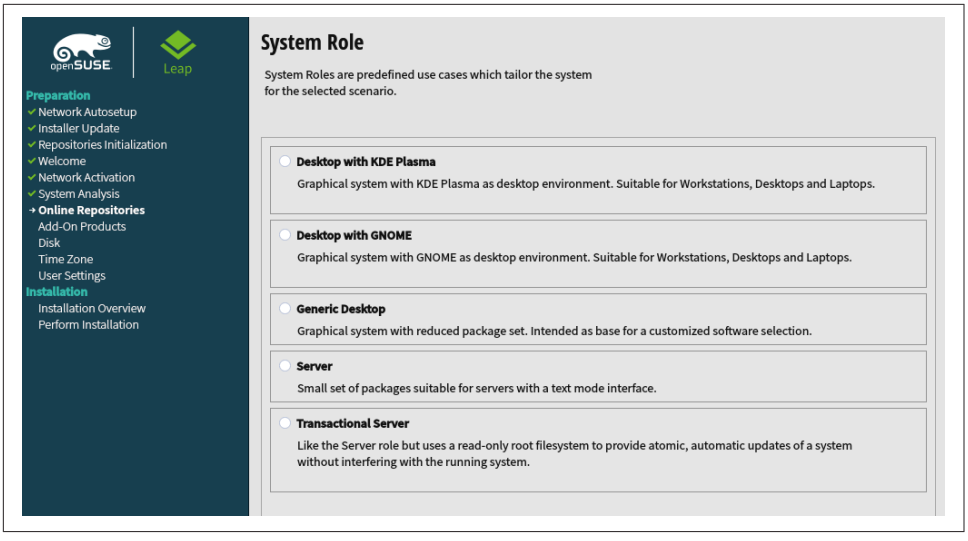


Figure 1-18. openSUSE installation roles

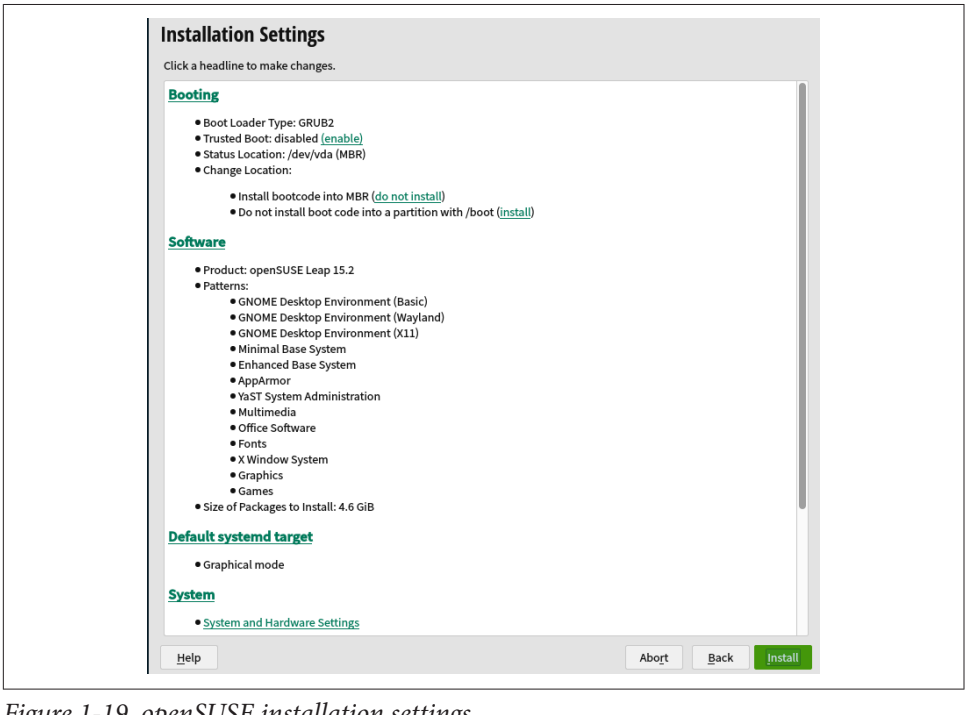


Figure 1-19. openSUSE installation settings

Click Software to open the package selection screen. This screen displays the openSUSE *patterns*, which are related groups of packages you can install with a single command. I like the Xfce desktop, so I am adding it to my installation (Figure 1-20).

Note the Details button at the bottom left. Click this to open a screen with multiple tabs for more fine-grained package selection (Figure 1-21). In this screen you will find a massive amount of information: the individual packages in each pattern, package groups, download repositories, installation summary, dependencies, and information on every package. Use the window on the right to select or deselect packages from each pattern. The installer will automatically resolve dependencies after your changes.

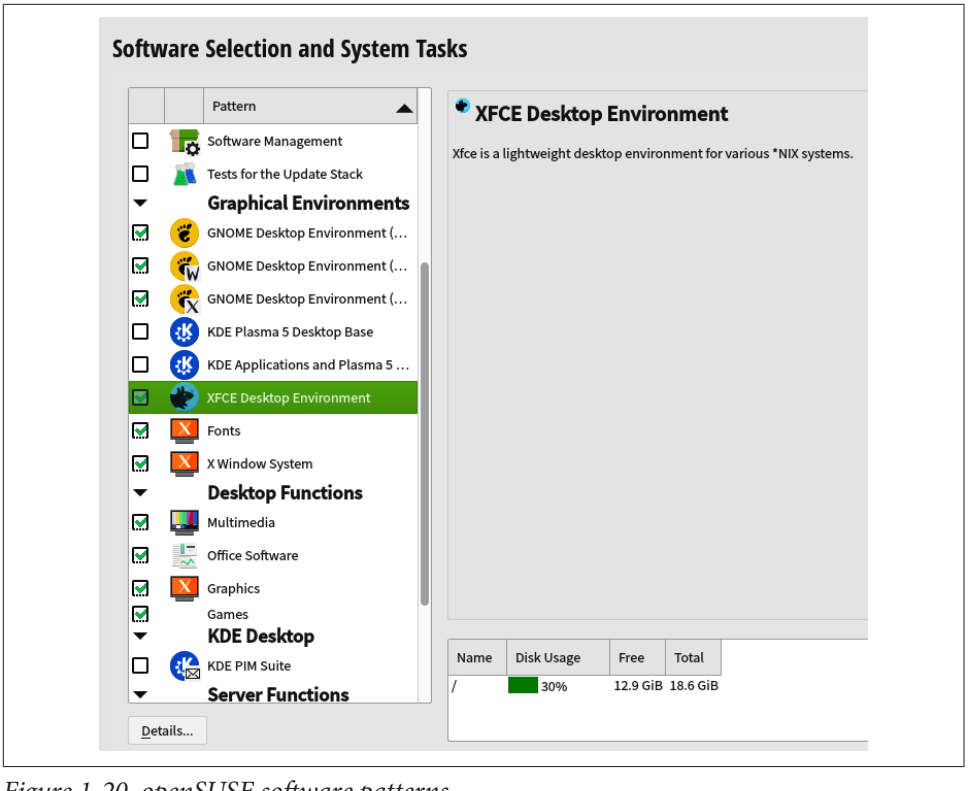


Figure 1-20. openSUSE software patterns

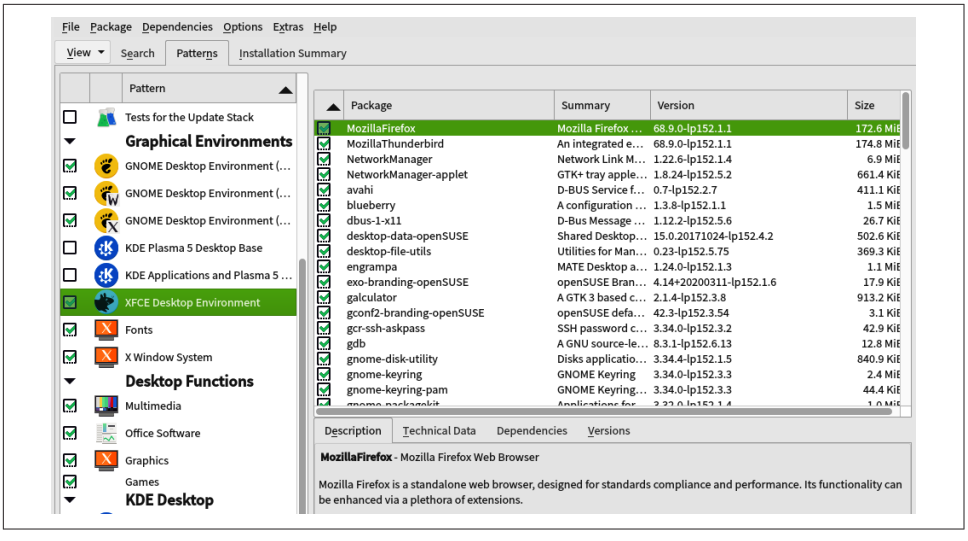


Figure 1-21. openSUSE individual package selection

When you are finished with software selection, you are returned to the Installation Summary screen, giving you another chance to change your installation settings. Click the green Next button to complete the installation.

Fedora Linux

The Fedora Linux Workstation and Server installers provide only partitioning options and no package customization. You need the 600 MB network installer image for a customizable installation, from Fedora Alternative Downloads. It is labeled as Fedora Server, but it provides complete package selection for any type of Fedora installation. Set up all your installation choices from the Installation Summary screen (Figure 1-22).

Take note of all the installation options on the Installation Summary screen: software selection, user creation, partitioning, keyboard and language, time zone, network, and hostname. Click Software Selection to open the screen for selecting the packages you want to install (Figure 1-23).

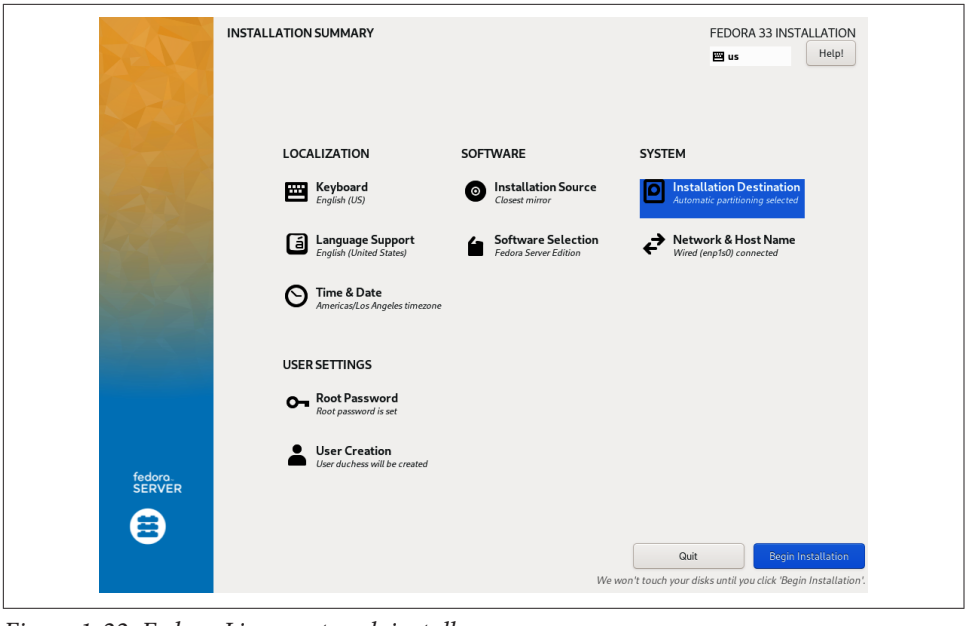


Figure 1-22. Fedora Linux network installer

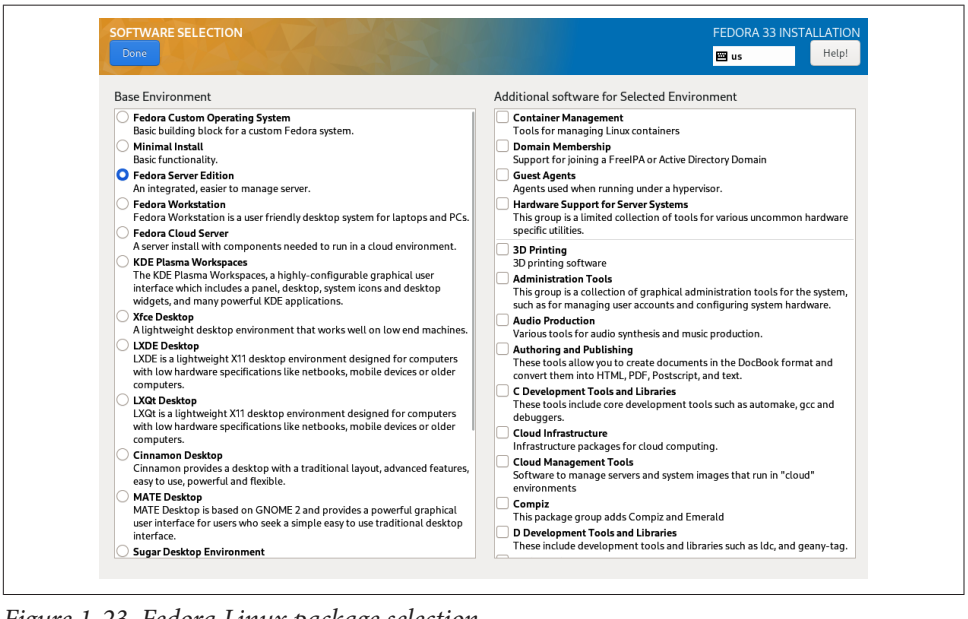


Figure 1-23. Fedora Linux package selection

When you are finished, click Done; you are returned to the Installation Summary screen. When you are finished setting up your installation, click Begin Installation, and the rest of the installation runs unattended.

Discussion

Whatever Linux you want to try, read its documentation and release notes. This is important information that saves a lot of aggravation. Also look for forums, mailing lists, and wikis to find help.

You may install as many desktop environments as you want, and then select the one you want to use when you log in. The button to select desktops is usually small and not obvious; for example, the default Ubuntu login screen (Figure 1-24) hides the desktop selector button until you click on a username. Xfce, Lxde, GNOME, and KDE are some of the popular graphical desktops. GNOME is the default on Ubuntu, openSUSE, and Fedora.

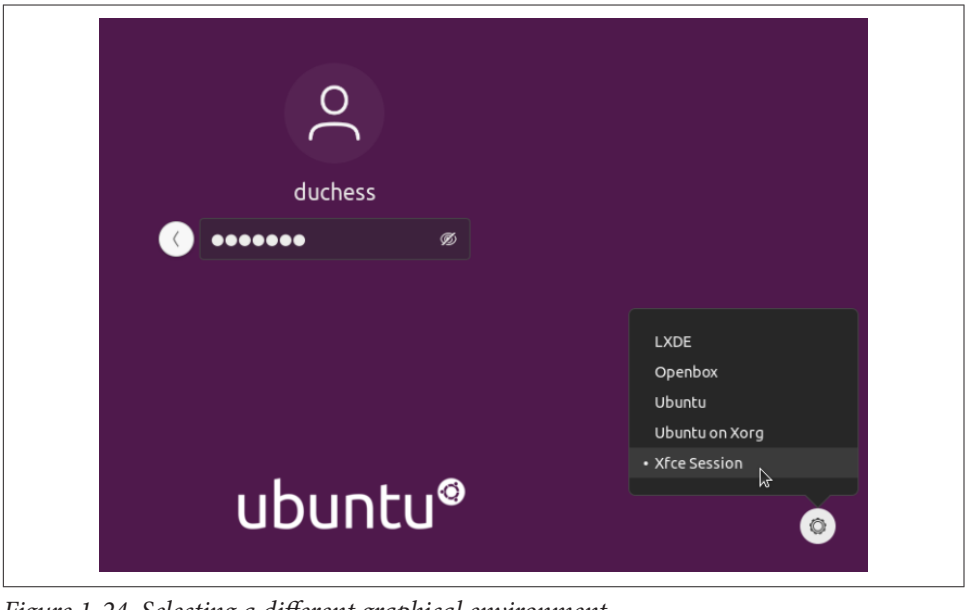


Figure 1-24. Selecting a different graphical environment

See Also

- openSUSE documentation
- SUSE Transactional Updates

1.11 Multibooting Linux Distributions

Problem

You want to install more than one Linux distribution on your computer in a multiboot setup, then select the one you want to run from your boot menu.

Solution

No worries, for you can install as many Linuxes as your hard disk (or disks) will hold. You must already have one Linux installed, and it must have a separate */boot* partition. Then the steps are:

1. Provide sufficient free disk space for the new Linux, which can be on the same hard disk as your existing Linux, or on a separate hard disk, either internal or external.
2. Make careful note of the partitions that belong to your first installed Linux, so that you do not accidentally overwrite or delete any partitions you want to keep.
3. Mount the */boot* partition in every new Linux that you install, and do not format it.
4. Boot to your installation medium, then configure the new installation to install on the free disk space.

The installer will automatically find your existing Linux installation and add the new Linux to the boot menu. After the installation is completed, you will see a boot menu like Figure 1-25, which has options to boot to Linux Mint or Ubuntu.

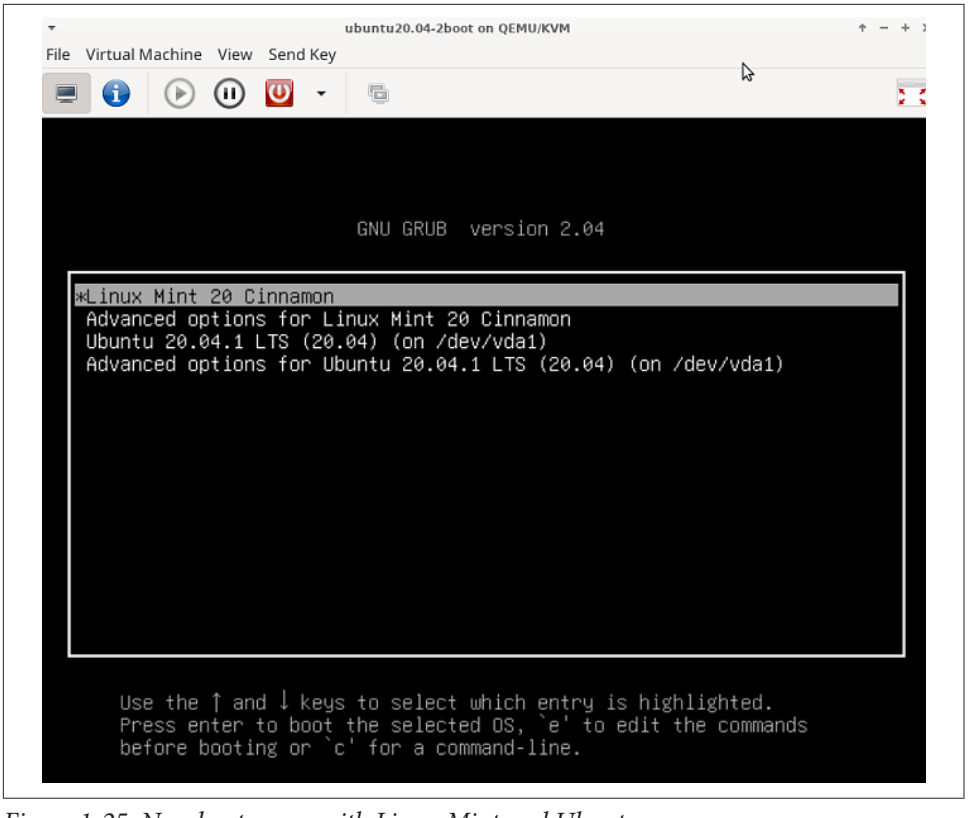


Figure 1-25. New boot menu with Linux Mint and Ubuntu

Discussion

If you need to free up space on your hard disk, you can shrink existing partitions safely (see Recipe 9.7) before you launch the installer. It is safest to do this on unmounted partitions, and some filesystems cannot be shrunk while they are mounted. Use SystemRescue to shrink partitions, see Recipe 19.12.

Most Linux installers are smart enough to recognize existing Linux installations and to offer the option to preserve them. In Figure 1-26 you see the Linux Mint installer, providing the options to take over your whole hard disk or to install next to Ubuntu without deleting it.

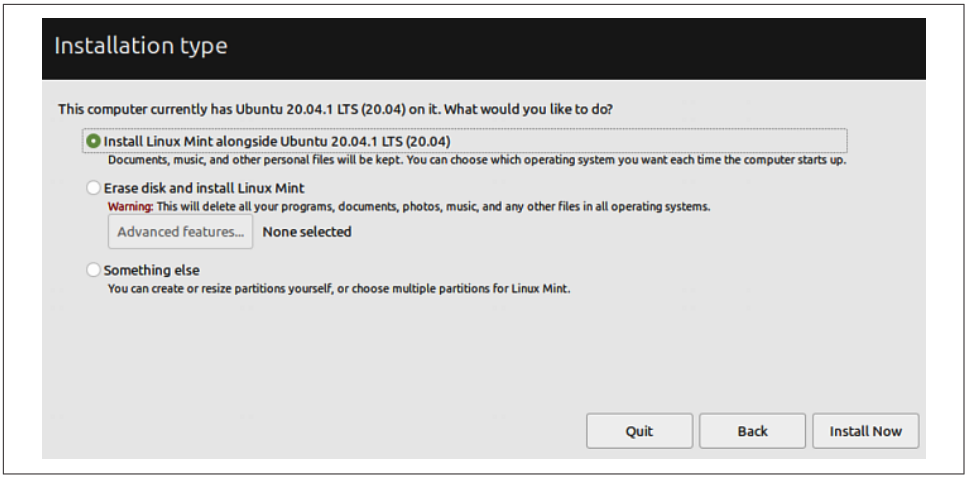


Figure 1-26. Installing Linux Mint next to Ubuntu

See Also

- Recipe 8.9
- Recipe 9.7
- Recipe 19.12
- The installation documentation for your Linux distribution

1.12 Dual-boot with Microsoft Windows

Problem

You want to dual-boot Linux and Windows on your computer.

Solution

Dual-booting Linux and Windows installs both systems on one computer, and then you choose the one you want to use from your boot menu at startup.

It is best to install Windows first, if it is not already installed, and install Linux second. Windows likes to control the bootloader, so installing Linux second allows Linux to take control.

As always, make sure you have fresh backups and Windows recovery media.

After Windows is installed, start your Linux installation. You will install Linux in whatever way you want: a simple installation or a customized installation where you set up partitioning and package selection. There is an important option specific to multibooting:

1. If you have a single hard drive, then the "Device for boot loader installation" is */dev/sda*.
2. If you have Windows on one hard disk and are installing Linux to a second hard disk, the "Device for boot loader installation" is the Linux disk. Use the device name, for example */dev/sdb*, not a partition name, like */dev/sdb1*.

In Figure 1-27, there are two hard drives, with Windows on */dev/sda* and Linux on */dev/sdb*.

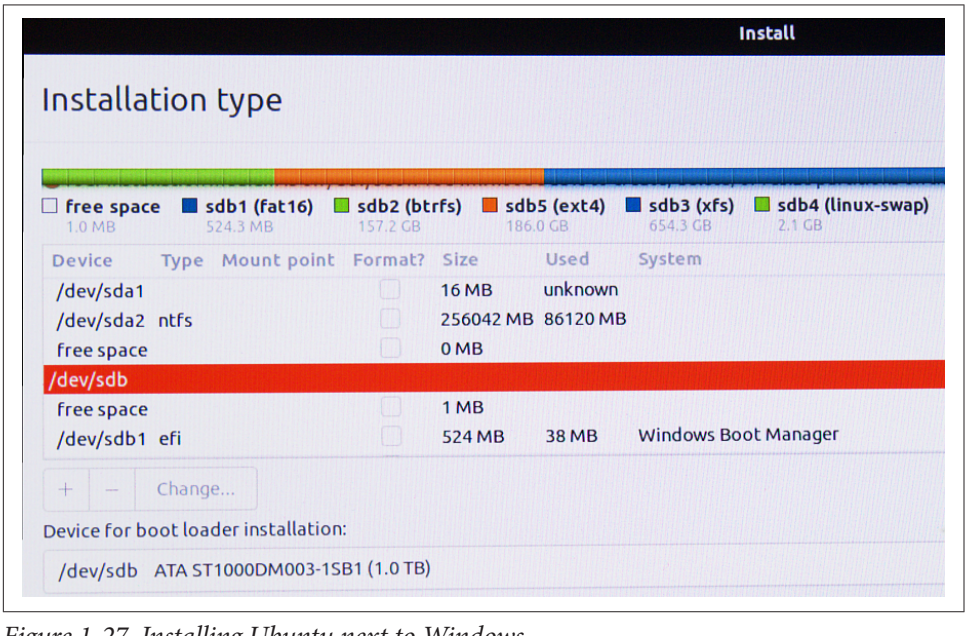


Figure 1-27. Installing Ubuntu next to Windows

Be very certain you are installing Linux to the correct location and not overwriting Windows. You may partition for Linux just as you would if you were installing it standalone, and again, be careful which partitions you change.

Once you have your partitioning set up and are satisfied with your installation configuration, go ahead and complete your Linux installation. After it is finished and you have rebooted, your GRUB menu will have entries for both systems (Figure 1-28).

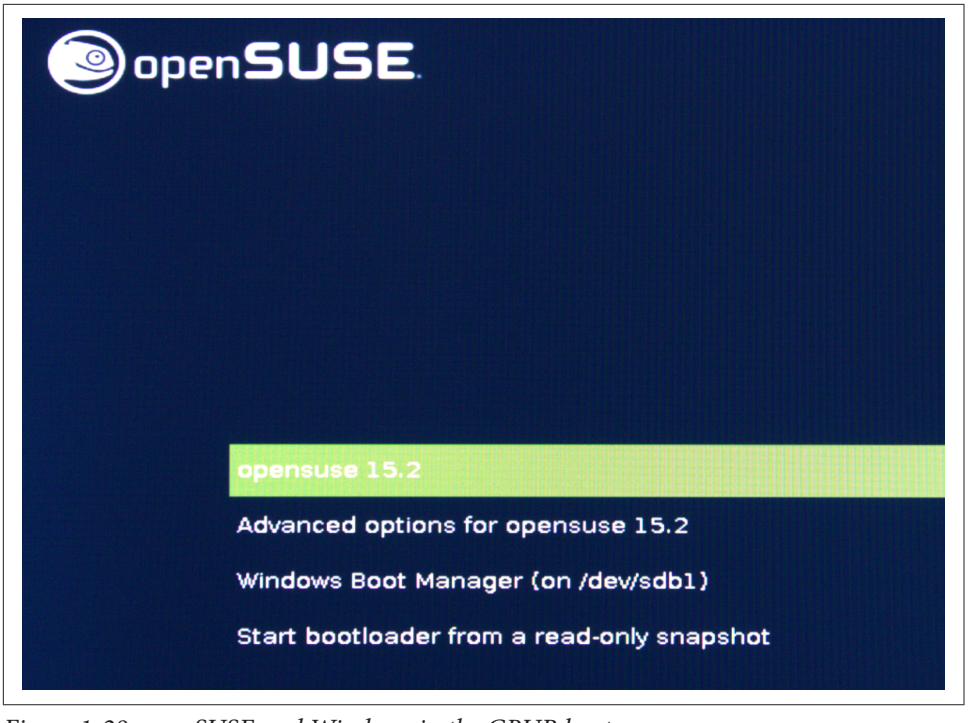


Figure 1-28. openSUSE and Windows in the GRUB boot menu

Discussion

You can install as many Linux and Windows systems on your system in a multiboot setup as you have room for on your hard drives.

There are other ways to run Linux and Windows on the same machine. Windows 10 includes the Windows Subsystem for Linux 2 (WSL 2), which runs supported Linux distributions in a virtual environment. You can run Windows in virtual machines on Linux if you have Windows installation media. Virtual machines are lovely because you can run multiple operating systems at the same time, though you need higher-end CPUs and lots of memory.

VirtualBox and QEMU/KVM/Virtual Machine Manager are good free virtual machine hosts that run on Linux.

See Also

- Windows Subsystem for Linux Documentation
- VirtualBox
- KVM

- Virtual Machine Manager
- QEMU

1.13 Recovering an OEM Windows 8 or 10 Product Key

Problem

You purchased a computer with Windows 8 or 10 preinstalled, and you can't find your product key.

Solution

Let Linux find it for you. Run the following command from a Linux system installed on the same computer with Windows, or from SystemRescue:

```
$ sudo cat /sys/firmware/acpi/tables/MSDM
MSDMU
DELL  CBX3
AMI
FAKEP-RODUC-TKEY1-22222-33333
```

And there it is on the last line.

If you can log in to Windows, run the following command in Windows to retrieve your product key:

```
C:\Users\Duchess> wmic path softwarelicensingservice get OA3xOriginalProductKey
OA3xOriginalProductKey
FAKEP-RODUC-TKEY1-22222-33333
```

Discussion

If you have no recovery media, Windows 10 is a free download. You will need your 25-digit OEM product key for a fresh installation.

See Also

- Download Windows 10

1.14 Mounting Your ISO Image on Linux

Problem

You have downloaded a Linux **.iso* file and are curious to see what it looks like after it is unpacked. You could go ahead and create a bootable DVD or USB stick, and then inspect the files, but you really want to unpack it without copying it to another device.

Solution

Linux has a pseudodevice called the *loop* device. This makes your **.iso* image accessible like any other filesystem. Follow these steps to mount your **.iso* image file in a loop device.

First, create a mountpoint in your home directory, giving it whatever name you want. In the example it is called *loopiso*:

```
$ mkdir loopiso
```

Mount your **.iso* in this new directory. In the example it is a Fedora Linux installation image:

```
$ sudo mount -o loop Fedora-Workstation-Live-x86_64-34-1.2.iso loopiso
mount: /home/duchess/loopiso: WARNING: device write-protected, mounted read-only
```

See the mounted filesystem in a file manager (Figure 1-29).

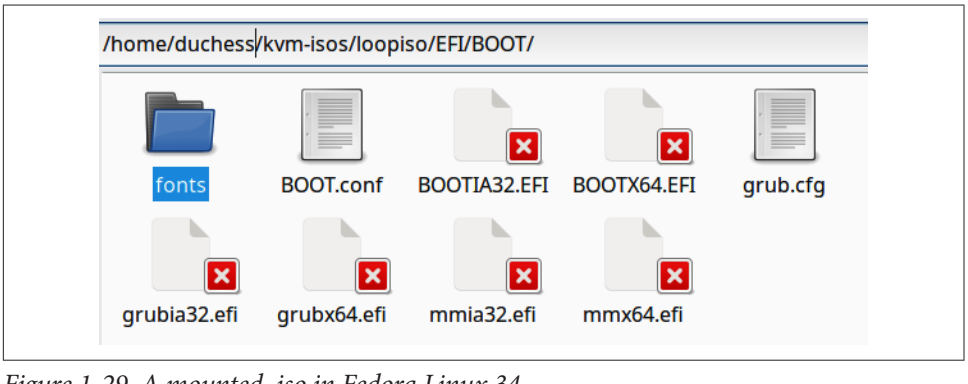


Figure 1-29. A mounted .iso in Fedora Linux 34

You can enter the directories and read the files. You won't be able to edit any files because they are mounted read-only.

When you are finished, unmount it:

```
$ sudo umount loopiso
```

Discussion

The loop device maps a regular file to a virtual partition, and you can set up a virtual filesystem in this file. If you want try to creating your own, a little bit of web searching will find a lot of how-tos. Start with *man 8 losetup*.

See Also

- *man 8 mount*
- *man 8 losetup*

CHAPTER 2

Managing the GRUB Bootloader

The *bootloader* is the software that loads your operating system after you power up your computer. The GRUB (GRand Unified Bootloader) bootloader is the most commonly used bootloader on Linux.

GRUB supports a number of useful features: boot multiple operating systems on a single PC, live configuration editing, themeable interface, and rescue modes. In this chapter you will learn about all of these.



GRUB versus GRUB 2

There are two major GRUB releases, legacy GRUB and GRUB 2. GRUB 2 is version 1.99 and up. Legacy GRUB ended at version 0.97 in 2005. A lot of GRUB how-tos still reference legacy GRUB and compare it with GRUB 2. In this chapter I'm not going to talk about legacy GRUB. It's been retired for a long time and has little relevance to using GRUB 2, so this chapter will focus exclusively on GRUB 2.

Some Linux distros use plain GRUB naming, some use GRUB 2. For example, Ubuntu has the */boot/grub/* directory and *grub-mkconfig* command, and Fedora calls them */boot/grub2/* and *grub2-mkconfig*. Check your filepaths and names. In this chapter I use the Ubuntu naming scheme, except in distro-specific examples.

Starting a computer hasn't changed all that much since UNIVAC was first built, back in the 1940s in the last millennium. Starting a computer is called *bootstrapping*, a reference to "pulling yourself up by your own bootstraps," which is impossible. The difficulty with a programmable computer is it needs software instructions to tell it what to do, but where will those instructions come from before the operating system is loaded?

The solution for the modern x86_64 PC architecture is to store the initial startup instructions on a chip on the motherboard and program the CPU with the address of these instructions. You could say the CPU is hardwired to receive the startup instructions. This address is the same on all x86_64 machines, and that is why you can mix and match motherboards and CPUs. (This address is called the *reset vector*, if you feel like doing some research.)

This is a simplified description of how it all works:

The first stage is launched when the system is powered up. The CPU fetches instructions from the BIOS/UEFI firmware, and then initializes CPU caches and system memory. When the system memory is initialized, the Power On Self-Test (POST) runs, testing the memory and testing connectivity with other hardware such as keyboard, mouse, display, and disk drives. You have probably noticed the LEDs on your keyboard and mouse lighting up and heard the noises from inside your computer's case as your disk drives are probed.

After the POST, the BIOS/UEFI firmware launches the second stage of startup and looks for the boot files on your hard disk. The GRUB bootloader loads the necessary files to launch your operating system and complete system startup.

When your boot screen appears (Figure 2-1), GRUB waits a configured amount of time for your input, usually 5–10 seconds, then boots the default if you do nothing. Navigate the boot menu with your arrow keys. When you press any key, it stops the countdown, and then you can explore your boot options at your leisure.

In Figure 2-1, the first entry boots the system. The next two entries open submenus with more boot options. When you are exploring submenus, press the Esc key to get back to the main menu.

Some Linux distros, such as Fedora and Ubuntu, do not display the boot screen when there is only one installed operating system. In this case, press the Shift key at startup to see the boot screen. There is a configuration option to always display the boot screen.

You may wish to customize the appearance and behavior of your GRUB menu by adjusting a few options in the GRUB configuration files.

If you prefer a graphical tool for customizing your GRUB menu, try GRUB Customizer (Figure 2-2). This is available in most Linux distributions as the *grub-customizer* package, except openSUSE, which has a GRUB module (labeled Boot Loader) in the YaST system configuration utility.

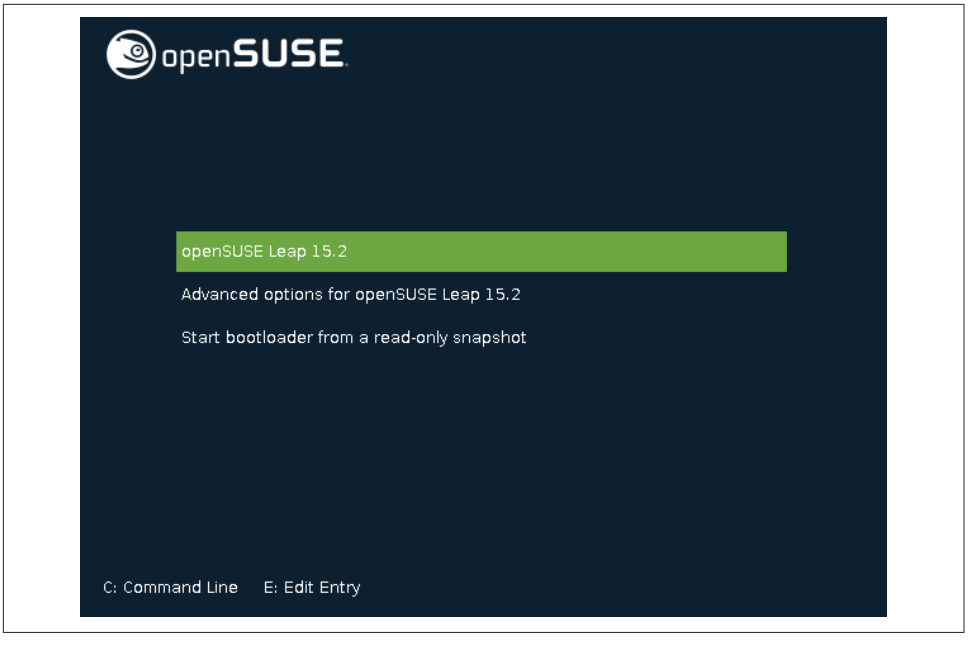


Figure 2-1. openSUSE GRUB boot screen

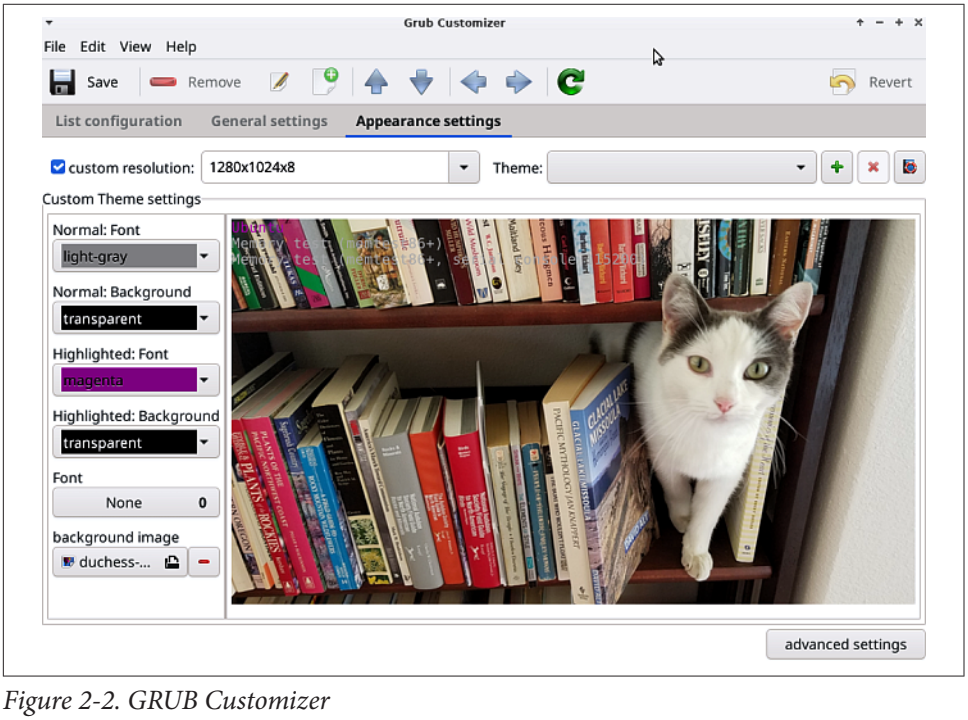


Figure 2-2. GRUB Customizer

2.1 Rebuilding Your GRUB Configuration File

Problem

Whenever you change your GRUB configuration, you need to rebuild it.

Solution

The command to rebuild your GRUB configuration varies. On Fedora and openSUSE, use this command:

```
$ sudo grub2-mkconfig -o /boot/grub2/grub.cfg
```

Some distros, such as Ubuntu, use:

```
$ sudo grub-mkconfig -o /boot/grub/grub.cfg
```

Ubuntu Linux also has a script that runs *grub-mkconfig*, *update-grub*:

```
$ sudo update-grub
```

Discussion

Some Linux distributions helpfully provide the correct command at the top of */etc/default/grub*.

Remember to always verify your correct filenames and paths when you edit your GRUB configuration because they vary on the different Linuxes.

See Also

- Your motherboard documentation, to learn about your system's BIOS/UEFI
- GNU GRUB Manual
- GRUB has multiple single-purpose man pages; run *man -k grub* to see all of them
- *info grub* or *info grub2*

2.2 Unhiding a Hidden GRUB Menu

Problem

Your favorite Linux distribution hides the GRUB menu when you have only one operating system installed on your computer, and you want it to appear every time you boot up.

Solution

Several Linux distros do this, including Ubuntu and Fedora. You can unhide your GRUB menu temporarily by pressing and holding the Shift key at startup.

Edit */etc/default/grub* to unhide it permanently with the following options:

```
GRUB_TIMEOUT="10"
GRUB_TIMEOUT_STYLE=menu
```

If the lines `GRUB_HIDDEN_TIMEOUT=0` and `GRUB_HIDDEN_TIMEOUT_QUIET=true` are in your file, comment them out.

After changing */etc/default/grub*, rebuild your GRUB configuration (Recipe 2.1).

Discussion

`GRUB_HIDDEN_TIMEOUT=0` means do not display the GRUB menu, while `GRUB_HIDDEN_TIMEOUT_QUIET=true` means do not display the countdown timer.

If you install another operating system in a multiboot configuration, then the GRUB menu should unhide itself.

See Also

- Recipe 2.1
- GNU GRUB Manual
- GRUB has multiple single-purpose man pages; run *man -k grub* to see all of them
- *info grub* or *info grub2*

2.3 Booting to a Different Linux Kernel

Problem

You are wondering about the extra entries in your GRUB menu that reference specific Linux kernel versions, like in Figure 2-3. You want to know what they are for and what to do with them.

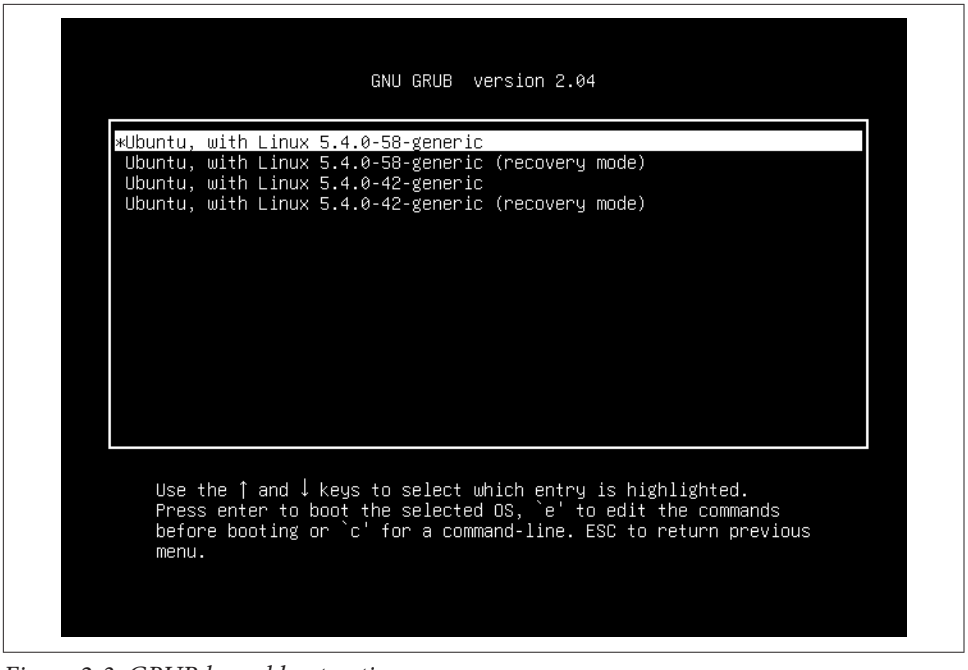


Figure 2-3. GRUB kernel boot options

Solution

Over time, as you update your Linux system, older Linux kernels are retained and added to your GRUB menu. This provides an easy way to boot to a known good older kernel if something goes awry with a newer kernel. You don't have to keep the older kernels and can remove them with your package manager.

Discussion

In olden times kernel updates were a big deal, because they often meant bug fixes, additional support for hardware, such as video, network, and audio interfaces, and support for software features, such as proprietary file formats and new protocols. The rate of change was rapid, and it was not unusual for a new kernel to not work correctly, so keeping the option to boot to an older kernel was routine. These days such troubles are less common, and kernel updates are generally undramatic.

See Also

- GNU GRUB Manual
- The Linux Kernel Archives

2.4 Understanding GRUB Configuration Files

Problem

You know that configuring GRUB is done a little differently than most programs, and you want to know where the GRUB configuration files are and which ones you use to manage GRUB.

Solution

GRUB configuration files are in */boot/grub/*, */etc/default/grub*, and */etc/grub.d/*. The GRUB configuration is complex, with many scripts and modules.



GRUB versus GRUB 2

Remember, as discussed in the introduction to this chapter, some Linux distros use plain GRUB naming, and some use GRUB2 in filenames and commands. In this chapter I use plain GRUB naming, except in distro-specific examples.

/etc/default/grub is for configuring the appearance of the GRUB menu you see at startup, such as hiding or showing the boot menu, applying themes and background images, menu timeout, and kernel options.

The files in */etc/grub.d/* support more complex configurations, and */boot/grub/* stores image and theme files for customizing the appearance of your GRUB menu.

The main GRUB configuration file is */boot/grub/grub.cfg*, which GRUB reads at startup. You do not edit this file because it is built from */etc/grub.d/* and */etc/default/grub*; every time you make configuration changes you must rebuild the GRUB configuration.

The GRUB configuration is automatically rebuilt when you install any updates that affect the boot process, such as installing newer kernels and removing older kernels.

Discussion

If you are interested in scripting, studying the GRUB files is an excellent course in organizing large numbers of interdependent scripts.

The files in */etc/grub.d/* are called *drop-in* files. Rather than dealing with a giant single configuration file, each drop-in file contains a configuration for a specific task. These files are numbered in the order that GRUB should read them, with lower numbers indicating higher priority. The following example is from Fedora 32:

```
$ sudo ls -C1 /etc/grub.d/
00_header
```

```
01_users
08_fallback_counting
10_linux
10_reset_boot_success
12_menu_auto_hide
20_linux_xen
20_ppc_terminfo
30_os-prober
30_uefi-firmware
40_custom
41_custom
backup
README
```

Each of these files is a script, and each must have the executable bit set. You can disable any of them by clearing the executable bit, like this:

```
$ sudo chmod -x 20_linux_xen
```

Re-enable a script by adding the executable bit:

```
$ sudo chmod +x 20_linux_xen
```

See Also

- GNU GRUB Manual
- GRUB has numerous man pages; run *man -k grub* to see all of them
- *info grub* or *info grub2*
- Chapter 6

2.5 Writing a Minimal GRUB Configuration File

Problem

You want to write the most minimal working GRUB configuration.

Solution

This is the most basic */etc/default/grub* file, with only the necessary entries to boot a Linux system and show the GRUB menu. The following example is for openSUSE Leap 15.2:

```
# If you change this file, run 'grub2-mkconfig -o /boot/grub2/grub.cfg'
# afterwards to update /boot/grub2/grub.cfg.

GRUB_DEFAULT=0
GRUB_TIMEOUT=10
GRUB_TIMEOUT_STYLE=menu
```

Remember Figure 2-1? Figure 2-4 is the same system, but with a minimal GRUB configuration.



Figure 2-4. Minimal GRUB menu

There are more options you can try, such as different ways to set the default boot option, change background images and themes, change the colors, and change the screen resolution. See the Discussion to learn about these.

Discussion

There are numerous options you can use in */etc/default/grub*, and you can ignore most of them. These are the options that I think are the most useful:

`GRUB_DEFAULT=`
: Sets the default boot entry. The boot entries are counted from 0 in *grub.cfg*, but not numbered. How do you know what numbers each boot option has? There is no obvious way to figure this out; you have to count your boot options manually. Count the "menuentry" sections to figure out the numbering. A menu entry looks like this:

```
menuentry 'openSUSE Leap 15.2'  --class opensuse --class gnu-linux
  --class gnu --class os
menuentry_id_option 'gnulinux-simple-102a6fce-8985-4896-a5f9-e5980cb21fdb' {
        load_video
        set gfxpayload=keep
        insmod gzio
        [...]
```

: Or use the *awk* command to list them for you, like this example for Ubuntu 20.04:

```
$ sudo awk -F\' '/menuentry / {print i++,$2}' /boot/grub/grub.cfg
0 Ubuntu
1 Ubuntu, with Linux 5.8.0-53-generic
2 Ubuntu, with Linux 5.8.0-53-generic (recovery mode)
3 Ubuntu, with Linux 5.8.0-50-generic
4 Ubuntu, with Linux 5.8.0-50-generic (recovery mode)
5 UEFI Firmware Settings
```

You probably don't want to use a recovery mode entry as the default, or a memory test, though it doesn't hurt anything if you do. UEFI Firmware Settings is a shortcut to your system's BIOS/UEFI.

`GRUB_TIMEOUT=10`
: Sets the number of seconds the GRUB menu waits before booting the default, and `GRUB_TIMEOUT_STYLE=menu` displays the menu during the countdown. `GRUB_TIMEOUT=0` boots immediately without displaying the menu, and `GRUB_TIMEOUT=-1` disables automatic booting and waits for the user to select a boot entry.

`GRUB_DEFAULT=saved`
: Together with `GRUB_SAVEDEFAULT=true`, `GRUB_DEFAULT=saved` makes the last menu entry that you booted the default for the next boot.

`GRUB_CMDLINE_LINUX=`
: Adds Linux kernel options for all menu entries.

`GRUB_CMDLINE_LINUX_DEFAULT=`
: Passes kernel options only to the default menu entries. `GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"` is a common default option that disables the verbose output at startup and displays a graphical splash screen. Figure 2-5 shows what the verbose output looks like. If you have `GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"` configured, you can see this output without changing your configuration by pressing the Esc key during startup.

`GRUB_TERMINAL=gfxterm`
: Sets the your GRUB screen to a graphical mode that supports colors and images. `GRUB_TERMINAL=console` disables graphical mode.

`GRUB_GFXMODE=`
: Sets the screen resolution for the graphical mode, for example, `GRUB_GFXMODE=1024x768`. Run the *set pager=1* command, and then run *videoinfo* from your GRUB command line to see your supported modes (Figure 2-6). *set pager=1* lets you use the arrow keys to page up and down long command output. `GRUB_GFXMODE=auto` calculates a reasonable default.

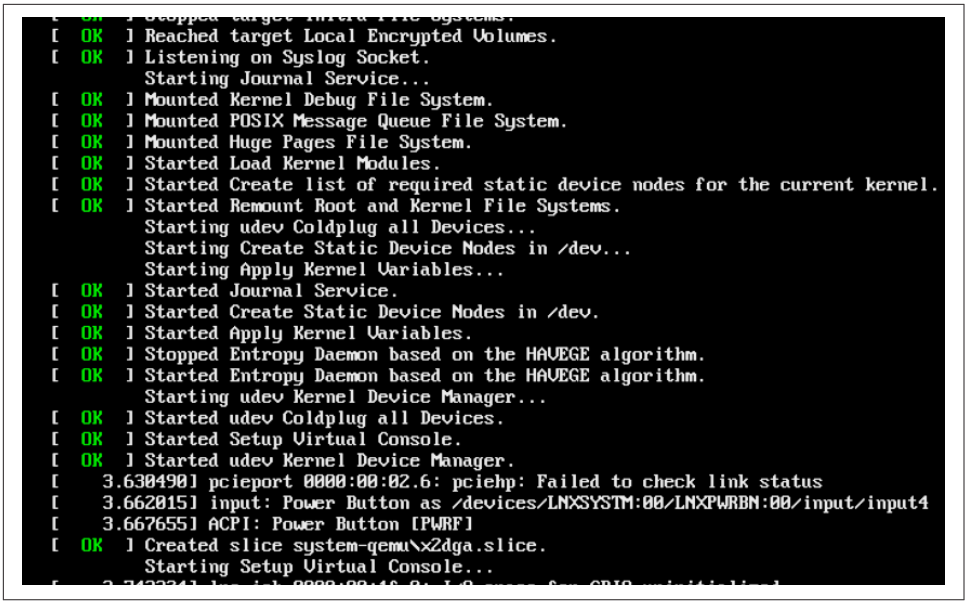


Figure 2-5. Startup messages

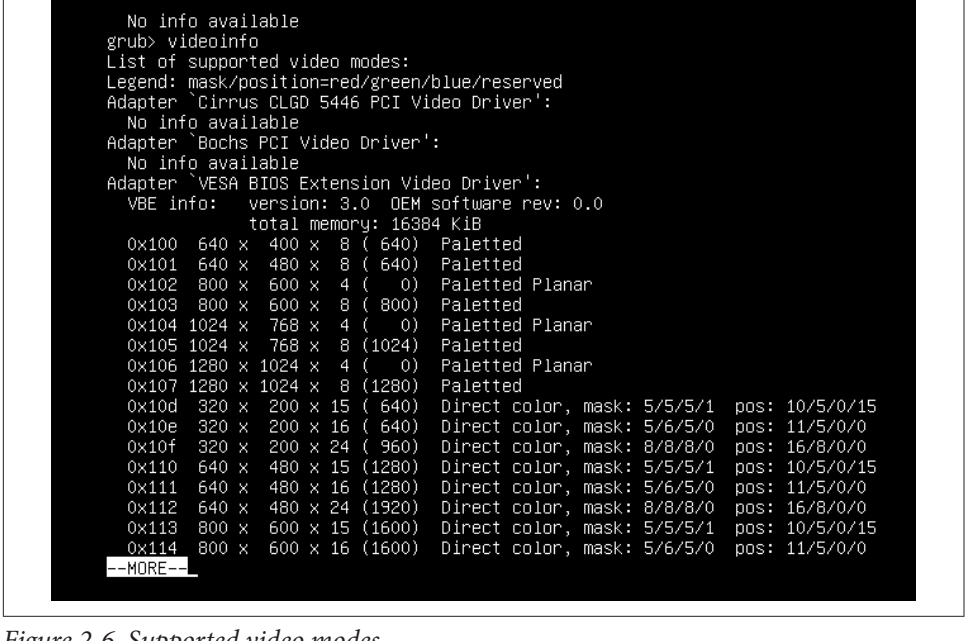


Figure 2-6. Supported video modes

`GRUB_BACKGROUND=`
: Sets a background image on the GRUB menu, using the image of your choice (see Recipe 2.6).

`GRUB_THEME=`
: Decorates your GRUB menu with a complete theme (see Recipe 2.8).

See Also

- Recipe 2.6
- Recipe 2.8
- GNU GRUB Manual
- GRUB has multiple single-purpose man pages; run *man -k grub* to see all of them
- *info grub* or *info grub2*

2.6 Setting a Custom Background for Your GRUB Menu

Problem

You don't care for the appearance of your GRUB menu, and you want to pretty it up.

Solution

You need an image in PNG, 8-bit JPG, or TFA format. It can be any size, and GRUB will scale it to fit. In the following example, a photo of Duchess enjoying the bookshelves graces our GRUB menu.

Copy your image to */boot/grub/*, and add the full filepath of your image to */etc/default/grub*. The photo of Duchess is */boot/grub/duchess-books.jpg*:

```
GRUB_BACKGROUND="/boot/grub/duchess-books.jpg"
```

If there is a `GRUB_THEME=` line, make sure it is commented out, then rebuild your GRUB configuration (Recipe 2.1).

You should see a line like "Found background: /boot/grub/duchess-books.jpg" in the output of your rebuild command. If you do not see this, there is an error in your configuration.

When it looks correct, rebuild, reboot, and enjoy your new GRUB menu background (Figure 2-7).



Figure 2-7. Duchess the literacy cat gracing the GRUB menu

The fonts in the example are barely readable, so jump ahead to Recipe 2.7 to learn how to change their colors.

Discussion

You may use any image on your system; it does not have to be in */boot/grub/*. Putting your image files in */boot/grub/* keeps all your GRUB customizations in one place and makes them available to all installed Linux systems on a multiboot setup.

See Also

- GNU GRUB Manual
- GRUB has multiple single-purpose man pages; run *man -k grub* to see all of them
- *info grub* or *info grub2*

2.7 Changing Font Colors in the GRUB Menu

Problem

Your new background is lovely (Figure 2-7), but your fonts are barely visible, and you need to change the colors so you can read your GRUB menu.

Solution

This is fun because you can quickly preview colors from the GRUB command line. Then, when you know what colors you want, edit */etc/default/grub* and create a new file in */etc/grub.d/* to load your colors, then rebuild */boot/grub/grub/cfg*. Reboot to enjoy your background image with pretty colored fonts.

Start up your computer, and when the GRUB menu appears, press C to open the GRUB command line (Figure 2-8).



Figure 2-8. The GRUB command line

The following two commands set the colors in Figure 2-9:

```
grub> menu_color_highlight=cyan/blue
grub> menu_color_normal=yellow/black
```

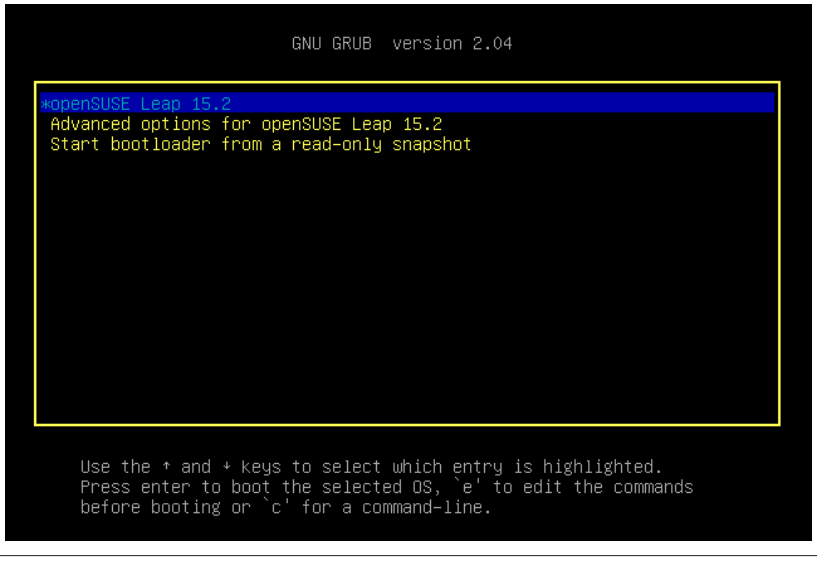


Figure 2-9. Setting GRUB menu colors from the GRUB command line

You may set and test each color pair one pair at a time. From the GRUB menu, press C to open the GRUB command shell. Type your command (sorry, no copy-paste), press Enter, then press Esc to return to the menu to see how it looks. You can list your previous commands with the up and down arrow keys, and edit and reuse them rather than retyping everything.

You must specify two colors in this order: foreground/background. All the colors are solid, with no transparency, with one exception: when you select *black* as a background color it is transparent. That is why `menu_color_normal=` must have black as the background color, when you have a background image. If you use any other color, your image will be covered by the background color. The `menu_color_highlight=` background color only applies to whatever line is currently selected.

When you figure out your colors, make them permanent. Boot up and create a new script in */etc/grub.d/*. In the following example, it is called *07_font_colors*. Copy this exactly:

```
#!/bin/sh

        if [ "x${GRUB_BACKGROUND}" != "x" ] ; then
                if [ "x${GRUB_COLOR_NORMAL}" != "x" ] ; then
                echo "set color_normal=${GRUB_COLOR_NORMAL}"
                fi

                if [ "x${GRUB_COLOR_HIGHLIGHT}" != "x" ] ; then
                echo "set color_highlight=${GRUB_COLOR_HIGHLIGHT}"
                fi
        fi
```

Then make it executable:

```
$ sudo chown +x 07_font_colors
```

Now add these lines to your */etc/default/grub* file, using your colors:

```
export GRUB_COLOR_NORMAL="yellow/black"
export GRUB_COLOR_HIGHLIGHT="cyan/blue"
```

Rebuild your GRUB configuration (Recipe 2.1) and reboot to see if it worked.

Discussion

See Recipe 2.4 to learn about the files in */etc/grub.d/* and why the filenames must start with numbers. In my experience it doesn't matter if your fonts script starts early or late, but if it doesn't work for you, try changing the priority. Make sure it is executable. The options are as follows:

- *menu_color_highlight* controls the colors of the highlighted lines inside the menu box.
- *menu_color_normal* controls the colors of the not-highlighted lines.

Use these colors exactly as they are written in Table 2-1, all lowercase and with this exact spelling.

Table 2-1. GRUB color options

Color options			
black	dark-gray	light-green	magenta
blue	green	light-gray	red
brown	light-cyan	light-magenta	white
cyan	light-blue	light-red	yellow

See Also

- GNU GRUB Manual
- GRUB has multiple single-purpose man pages; run *man -k grub* to see all of them
- *info grub* or *info grub2*
- Recipe 2.4

2.8 Applying a Theme to Your GRUB Menu

Problem

You like prettying up your GRUB menu, and you want to know if there are themes for the GRUB menu and how to install them.

Solution

You are in luck, for there are many themes for GRUB. Start by using your package manager with the **`grep`** command to search for package names. This example is for Ubuntu Linux:

```
$ apt search theme | grep grub
```

Using *theme* | *grep grub* to filter your results will find all relevant packages, like *grub-theme-breeze*, *grub2-themes-ubuntu-mate*, and *grub-breeze-theme*. Install themes just as you would any package.

Discussion

Your new theme should be installed in */boot/grub/themes*. Find your new theme, for example */boot/grub/themes/ubuntu-mate*, and look for the *theme.txt* file. Enter the full path in */etc/default/grub*, like this example for the *ubuntu-mate* theme:

```
GRUB_THEME=/boot/grub/themes/ubuntu-mate/theme.txt
```

Be sure to comment out any other configuration lines pertaining to appearance that you may have, such as `GRUB_BACKGROUND=`, any custom font colors, and any other themes. Then rebuild your GRUB configuration (Recipe 2.1). You should see a line like "Found theme: /boot/grub/themes/ubuntu-mate/theme.txt" in your command output.

If all goes well, reboot and you will be rewarded with a screen like Figure 2-10. If it does not display correctly, recheck your configuration and commands.

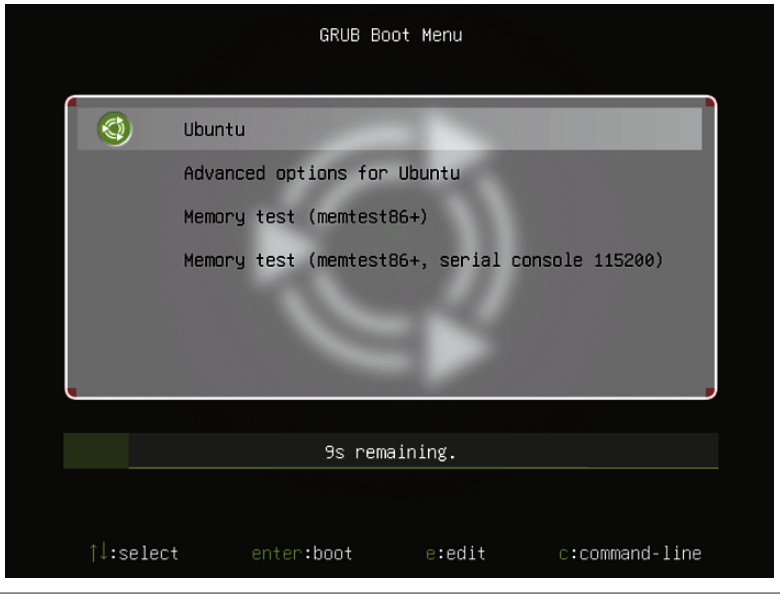


Figure 2-10. The Ubuntu MATE GRUB theme

See Also

- GNOME Themes
- KDE Themes
- GNU GRUB Manual

2.9 Rescuing a Nonbooting System from the grub> Prompt

Problem

When you start your system, it stops at a GRUB prompt, `grub>`, and does not boot. You need to know how to boot your system and then repair your configuration.

Solution

When the boot process stops at the `grub>` prompt (Figure 2-11), this means it found */boot/grub/* but cannot find the root filesystem.



Figure 2-11. The GRUB command shell

You need to find the root filesystem, the Linux kernel, and its matching *initrd* file. When you are in the GRUB command shell, the entire filesystem is open to you.

The first command you should run is to invoke the pager, so that you can page up and down long output:

```
grub> set pager=1
```

List your disks and partitions. GRUB has its own way of identifying hard disks and partitions. It numbers disks from 0, partitions from 1, and labels all hard disks as *hd*. On a running Linux system, hard disks are identified as */dev/sda*, */dev/sdb*, and so on.

In the following example, GRUB lists two hard disks, *hd0* and *hd1*, which are the same as */dev/sda* and */dev/sdb*. *hd0,gpt5* is the same as */dev/sda5*, and *hd1,msdos1* is the same as */dev/sdb1*:

```
grub> ls
(hd,0) (hd0,gpt5) (hd0,gpt4) (hd0,gpt3) (hd0,gpt2) (hd0,gpt1)
(hd1) (hd1,msdos1)
```

This output shows that *hd0* has a *gpt* partition table, and *hd1* has the old-fashioned *msdos* partition table. It is not necessary to use the *gpt* and *msdos* labels when you are listing partitions and files.

GRUB tells you the filesystem types, universally unique identifiers (UUIDs), and other information on the partitions:

```
grub> ls (hd0,3)
  Partition hd0,3: filesystem type ext* - Last modification time 2021-12-29
  01:17:58 Tuesday, UUID 5c44d8b2-e34a-4464-8fa8-222363cd1aff - Partition start
  at 526336KiB -
Total size 20444160KiB
```

You need to find */boot*. Suppose you remember that it is in the root filesystem on the second partition; start looking there. A forward slash following the partition name means list all files and directories on the partition:

```
grub> ls (hd0,2)/
bin   dev  home  lib64   media  opt   root  sbin  sys  usr
boot  etc  lib   lost+found  mnt   proc  run   srv   tmp  var
```

All boot files are in the */boot* directory:

```
grub> ls (hd0,2)/boot
efi/ grub/ System.map-5.3.18-lp152.57-default config-5.3.18-lp152.57-default
initrd-5.3.18-lp152.57-default vmlinuz vmlinuz-5.3.18-lp152.57-default
sysctl.conf-5.3.18-lp152.57-default vmlinux-5.3.18-lp152.57-default.gz
```

Everything you need to boot your system is there. Set the root filesystem partition, kernel, and initrd image:

```
grub> set root=(hd0,2)
grub> linux /boot/vmlinuz-5.3.18-lp152.57-default root=/dev/sda2
grub> initrd /boot/initrd-5.3.18-lp152.57-default
grub> boot
```



Tab Completion

Just like the Bash shell, the GRUB command shell supports tab completion. This means you can start typing */boot/vml*, for example, then press the Tab key to autocomplete the line, or display a list of possibilities.

If there are multiple *vmlinuz* and *initrd* files, use the two with the newest matching version numbers. If all the commands are correct, your system will boot and you can fix your GRUB configuration (Recipe 2.11).

Discussion

When */boot* is in its own partition, you won't see any other directories because it is not in the root filesystem.

vmlinuz-5.3.18-lp152.57-default is the compressed Linux kernel.

initrd-5.3.18-lp152.57-default is the initial ramdisk, a temporary root filesystem used only to start up your system.

Boot failures are caused by corrupted files; adding, removing, or moving hard disks; installing or removing operating systems; or repartitioning. If you can't get to a GRUB prompt, see Chapter 19 to learn how to rescue your system with SystemRescue.

You can practice using the `grub>` shell by pressing C when your GRUB menu appears. This is safe because changes you make do not survive a reboot.

See Also

- GNU GRUB Manual

2.10 Rescuing a Nonbooting System from the grub rescue> Prompt

Problem

When you start your system, it stops at a GRUB prompt, `grub rescue>`, and does not boot. You need to know how to boot your system and then repair your configuration.

Solution

The `grub rescue>` prompt (Figure 2-12) is the rescue shell, which means GRUB could not find */boot*. Not to worry, you can find it from the GRUB prompt, boot your system, and then make a permanent fix.

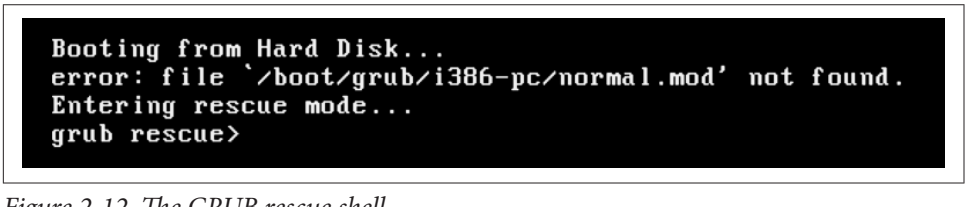


Figure 2-12. The GRUB rescue shell

List your partitions:

```
grub rescue> ls
(hd0) (hd0,gpt5) (hd0,gpt4) (hd0,gpt3) (hd0,gpt2) (hd0,gpt1)
(hd1) (hd1, msdos1)
```

At this point there is no tab-completion or paging, so you have to type everything.

GRUB tells you the filesystem types, UUIDs, and other information on the partitions:

```
grub rescue> ls (hd0,3)
    Partition hd0,3: filesystem type ext* - Last modification time 2021-12-29
    01:17:58
Tuesday, UUID 5c44d8b2-e34a-4464-8fa8-222363cd1aff - Partition start at
526336KiB -
Total size 20444160KiB
```

If you don't know which partition contains */boot*, you will have to list the files and directories in each one until you find it. You do not have to use the *gpt* and *msdos* labels. A forward slash following the device name means list all files and directories:

```
grub rescue> ls (hd0,2)/
bin   dev  home  lib64   media  opt   root  sbin  sys  usr
boot  etc  lib   lost+found  mnt   proc  run   srv   tmp  var
```

Hurrah, there it is in the root filesystem. List the files in */boot*:

```
grub rescue> ls (hd0,2)/boot
efi/ grub/ System.map-5.3.18-lp152.57-default config-5.3.18-lp152.57-default
initrd-5.3.18-lp152.57-default vmlinuz vmlinuz-5.3.18-lp152.57-default
sysctl.conf-5.3.18-lp152.57-default vmlinux-5.3.18-lp152.57-default.gz
```

There are a few additional commands for `grub rescue>`. You have to tell it where */boot/grub* is, and then load the *normal* and *linux* kernel modules, which are in */boot/grub/i386-pc* (along with many other kernel modules that GRUB uses at startup). *normal* changes the boot mode from rescue to normal, and *linux* starts the system loader:

```
grub rescue> set prefix=(hd0,2)/boot/grub
grub rescue> set root=(hd0,2)
grub rescue> insmod normal
grub rescue> insmod linux
```

After loading *normal* and *linux*, you have tab completion. You could also turn on paging, *set pager=1*, to enable using the arrow keys to scroll to your previous commands. Now tell GRUB where to find the *kernel* and *initrd* file:

```
grub> linux /boot/vmlinuz-5.3.18-lp152.57-default root=/dev/sda2
grub> initrd /boot/initrd-5.3.18-lp152.57-default
grub> boot
```

If there are multiple *vmlinuz* and *initrd* files, use the two with the newest matching version numbers. If all the commands are correct, your system will boot and you can fix your GRUB configuration (Recipe 2.11).

Discussion

When */boot* is on its own partition, you won't see any other directories because it is in its own filesystem.

See Also

- GNU GRUB Manual

2.11 Reinstalling Your GRUB Configuration

Problem

You were able to boot your system from the GRUB prompt, and now you need to know how to make a permanent repair.

Solution

Check your GRUB configuration carefully for errors. When it looks correct, rebuild your GRUB configuration (Recipe 2.1). Then you need to reinstall GRUB. In the following example, it is reinstalled to */dev/sda*:

```
$ sudo grub-mkconfig -o /boot/grub/grub.cfg
$ sudo grub-install /dev/sda
```



Using the Correct Rebuild Command

As discussed in Recipe 2.1 and "GRUB versus GRUB 2" on page 37, you must check your filepaths to ensure you are using the correct rebuild command.

Be sure to install it to the correct disk, if you have more than one, and use only the device name (for example, */dev/sda*) and not a partition (such as */dev/sda1*).

Discussion

Be sure to have good current backups. If your rescue efforts fail, try reinstalling GRUB from SystemRescue (Recipe 19.9).

See Also

- GNU GRUB Manual
- Chapter 19

CHAPTER 3

Starting, Stopping, Restarting, and Putting Linux into Sleep Modes

In this chapter you will learn several ways to stop, start, and restart a Linux system, and how to manage sleep modes. You will learn both the legacy commands and the new systemd commands.

You will also learn how to set up automated startups and shutdowns. An automated shutdown is nice to remind you to stop working, and you don't have to remember to shut off your computer for the night. You can set up automated wake-ups and shutdowns on a remote machine, so that you can access it during work hours without leaving it running all the time. If your users are watt wasters who don't shut off their computers, you can configure them to shut down during off hours.

The "three-key salute," Ctrl-Alt-Delete, is useful when you need to interrupt a startup and reboot, or reboot when a process or application is misbehaving. In graphical desktops you can remap the keys to a more convenient key combination.

There are a number of legacy shutdown commands that have accumulated over the decades, with a lot of overlapping functionality: *shutdown*, *halt*, *poweroff*, and *reboot*. The *shutdown* command provides the useful options of timed shutdowns, with warnings to all logged-in users. These commands are useful in scripts, in SSH sessions, and anytime you are working from the command line.



Root Privileges Not Always Required

In olden times, root privileges were needed to run the shutdown commands. This is changing, and on many modern Linux distributions, root privileges are not needed for these commands. The examples in this chapter are for normal, unprivileged users. If your particular Linux requires root permissions, it will tell you.

These permissions are controlled by Polkit (formerly PolicyKit) on modern Linux distributions. See *man 8 polkit* to learn more.

But that's not all, because in Linux distributions with systemd (Chapter 4), the classic old commands are not installed on the system. Intead, their names are symlinked to the *systemctl* command. You can see this with the *stat* command, like this example for *shutdown*:

```
$ stat /sbin/shutdown
  File: /sbin/shutdown -> /bin/systemctl
  Size: 14              Blocks: 0          IO Block: 4096   symbolic link
Device: 802h/2050d      Inode: 1177556     Links: 1
Access: (0777/lrwxrwxrwx)  Uid: ( 0/ root)   Gid: ( 0/ root)
```

The `File:` line displays a symbolic link to */bin/systemctl*. All of the legacy command names, */sbin/shutdown*, */sbin/halt*, */sbin/poweroff*, and */sbin/reboot*, are symlinked to */bin/systemctl*. The symlinks for the legacy command names are provided for backward compatibility. On Linux systems without systemd these symlinks do not exist, and these systems use the legacy executables.

On some Linux distros these symlinks are in */usr/sbin* rather than */sbin*. When you use the legacy command names, they behave the same way on systems with systemd and on systems without systemd.

The power buttons on your graphical desktop are configurable; you should be able to customize which buttons are visible and their location.

3.1 Shutting Down with systemctl

Problem

You want to use the *systemctl* commands to shut down and reboot your system.

Solution

Halt the system and power off the machine:

```
$ systemctl poweroff
```

Another way to halt the system and power off the machine:

```
$ systemctl shutdown
```

Reboot:

```
$ systemctl reboot
```

Halt the system without powering off the machine:

```
$ systemctl halt
```

Discussion

The *systemctl* shutdown commands do not have the many options that the legacy commands do, which does not matter all that much because there are a lot of redundant options in the legacy commands. There is one significant difference: *systemctl shutdown* lacks the timed shutdown options supported by the *shutdown* command (see Recipe 3.2).

See Also

- *man 8 systemd-halt.service*

3.2 Shutting Down, Timed Shutdowns, and Rebooting with the shutdown Command

Problem

You want to use timed shutdowns, for example, 10 minutes from now or at a specific time, and warn all logged-in users. Or you want to just shut down now with no frills.

Solution

The *shutdown* command works the same way, whether it is symlinked to *systemctl* or the legacy *shutdown* executable.

The following examples show how to shut down immediately, shut down a certain number of minutes from now, cancel a shut down, shutdown at a specific time, halt, and reboot.

Shut down immediately, with no notification to other logged-in users:

```
$ shutdown -h now
```

Shut down in 10 minutes with notifications:

```
$ shutdown -h +10
Shutdown scheduled for Sun 2021-05-23 11:04:43 PDT, use 'shutdown -c' to cancel.
```

Other users on the system may see this message, depending on which Linux they are using, and if they have a terminal open:

```
Broadcast message from duchess@client4 on pts/4 (Sun 2021-05-24 10:54:43 PDT):

The system is going down for poweroff at Sun 2021-05-24 11:04:43 PDT!
```

Cancel the shutdown:

```
$ shutdown -c
```

Logged-in users may see this message:

```
Broadcast message from duchess@client4 on pts/4 (Sun 2021-05-24 10:56:00 PDT):

The system shutdown has been cancelled
```

Create your own message:

```
$ shutdown -h +6 "Time to stop working and go outside to play!"
```

Rather than specifying minutes to shutdown, you may set a shutdown time in 24-hour hh:mm format. The following example shuts down the system at 10:15 P.M.:

```
$ shutdown -h 22:15
```

Reboot:

```
$ shutdown -r
```

Halt the system without powering off:

```
$ shutdown -H
```

Running *shutdown* with no options is equivalent to *shutdown -h +1*.

Discussion

shutdown sends messages only when you use the *-h* option, except for the *-h now* option, and when you use the *-k* option. These are called *wall* messages, which is short for "write to all logged-in users." Linux distributions vary in their support of this feature, so other users on your particular Linux may not see these messages.

- *--help* displays a summary of options.
- *-H*, *--halt* performs a clean shutdown but does not power off the machine; you must press and hold the power button to power off your machine.
- *-P*, *--poweroff* performs a clean shutdown and powers off the machine.
- *-r*, *--reboot* performs a clean shutdown and reboots the machine.

- *-k* sends a wall message without shutting down the machine.
- *--no-wall* disables wall messages.

See Also

- *man 8 shutdown*
- *man 1 wall*
- *man 8 systemd-halt.service*

3.3 Shutting Down and Rebooting with halt, reboot, and poweroff

Problem

You understand the *shutdown* command, and now you want to know what *halt*, *reboot*, and *poweroff* are for, and how to use them.

Solution

These are all pretty much the same.

halt performs a clean shutdown, stopping all services and processes and unmounting filesystems, but it does not power off the machine. After the *halt* command is finished, you must press and hold the machine's power button to complete the shutdown.

reboot performs a clean shutdown and restarts the system.

poweroff performs a clean shutdown and powers off the machine:

```
$ halt
$ reboot
$ poweroff
```

The *halt* and *poweroff* commands can reboot the system:

```
$ halt --reboot
$ poweroff --reboot
```

Discussion

If you're thinking this all looks a little weird and redundant, you're right. As software ages, cruft accumulates and never goes away. Linux has been around since 1991, and

started out as a free clone of Unix, which was born in 1969. That is a lot of years for the various contributors to tweak code and add their own favorite features.

halt and *poweroff* are the same commands and support the same options:

- *--help* displays a summary of options.
- *--halt* performs a clean shutdown but does not power off the machine (yes, *halt* and *halt --halt* do the same thing).
- *-p*, *--poweroff* performs a clean shutdown and powers off the machine (yes, *poweroff* and *poweroff --poweroff* do the same thing) .
- *--reboot* performs a clean shutdown and restarts the machine.
- *-f*, *--force* forces an immediate halt or poweroff. Shutdown of all running services is skipped, all processes are killed, and all filesystems are unmounted or mounted read-only. Run it twice to force an unclean shutdown; for example, *poweroff -f -f*, which you should use only when normal shutdown commands fail.
- *-w*, *--wtmp-only* does not perform a shutdown, but only writes an entry in */var/log/wtmp*.
- *-d*, *--no-wtmp* prevents writing a *wtmp* entry.

See Also

- *man 8 halt*
- *man 8 poweroff*
- *man 8 systemd-halt.service*

3.4 Sending Your System into Sleep Modes with systemctl

Problem

Your Linux system has systemd, and you want to use *systemctl* to manage system sleep modes.

Solution

systemctl provides these power saving modes: *suspend*, *hibernate*, *hybrid-sleep*, and *suspend-then-hibernate*.

Put your system into suspend mode:

```
$ systemctl suspend
```

This stores your current session in RAM and puts all hardware into a suspended state. Wake your system up by pressing any key, moving the mouse, or opening your laptop lid.

Put your system into hibernate mode:

```
$ systemctl hibernate
```

This stores your session on disk and powers off the machine. Wake it up by pressing the power button, and when it resumes, which can take a minute or two, your session will be restored where you left off.

Put your system into hybrid-sleep mode:

```
$ systemctl hybrid-sleep
```

This suspends your system to both memory and disk, and shuts off all devices except RAM. If your system RAM loses power, the system resumes from disk. Wake it up by pressing the power button.

Put your system into suspend-then-hibernate mode:

```
$ systemctl suspend-then-hibernate
```

suspend-then-hibernate first goes into suspend mode, then enters hibernation after the period of time specified by the HibernateDelaySec= setting in */etc/systemd/sleep.conf*. Wake it up by pressing the power button.

Discussion

See the Discussion in Recipe 3.9 for details on the different sleep states.

Your graphical desktop should have buttons for entering power saving modes and a graphical configuration tool for controlling events such as screen blanking, screen locking, power button, mouse, and laptop lid actions such as entering a sleep state or powering off.

Power management tends to work best on laptops and may not behave as expected on your Linux distribution. Power management is affected by your UEFI, CPU capabilities, udev, Advanced Configuration and Power Interface (ACPI), kernel compilation options, and possibly other devices and programs; so much depends on how your particular Linux has implemented power management. Check your Linux distribution documentation.

See Also

- *man 1 systemctl*
- *man 8 systemd-halt.service*

3.5 Rebooting Out of Trouble with Ctrl-Alt-Delete

Problem

You want a reliable method of rebooting that always works, even when you are having problems such as crashes and runaway processes.

Solution

The good old "three-key salute," Ctrl-Alt-Delete, was made for this. Press and hold these three keys in sequence, and they will override most problems and reboot your system. Ctrl-Alt-Delete is disabled in some Linux distributions, and you can change this.

Ctrl-Alt-Delete is controlled by systemd in the Linux console. See Recipe 3.6 to learn about managing Ctrl-Alt-Delete in systemd.

For systems without systemd, see the Discussion in this recipe.

Graphical environments have their own configuration tools for Ctrl-Alt-Delete, independent of systemd. For example, there is a keyboard configuration module in the Xfce4 Settings Manager (Figure 3-1); in GNOME, use the Keyboard Settings module in the GNOME Settings utility.

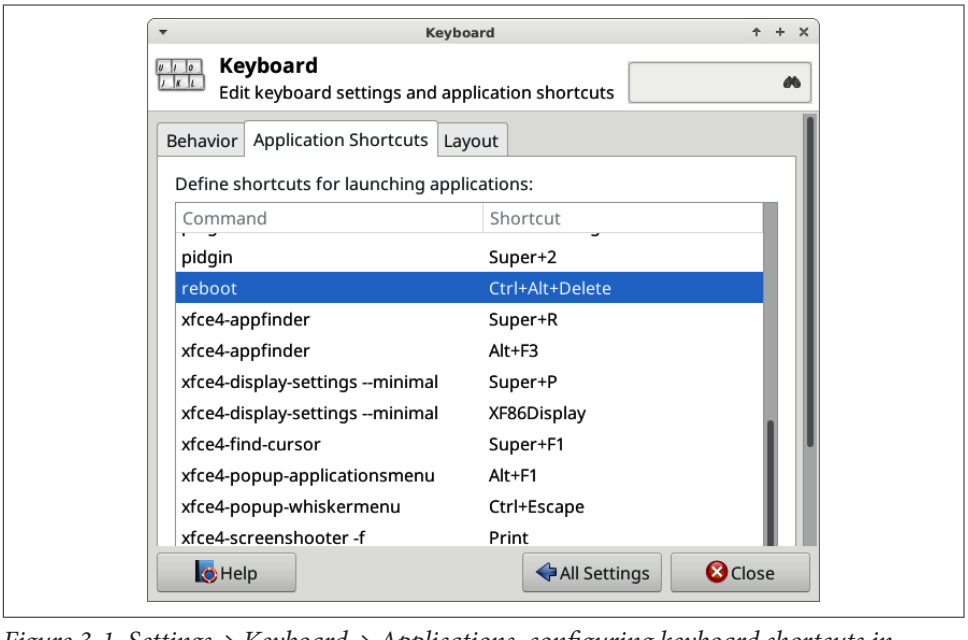


Figure 3-1. Settings → Keyboard → Applications, configuring keyboard shortcuts in Xubuntu

If you prefer a different key combination, use any keys you want. You could even use a single key, though that risks rebooting with an accidental key press.

Discussion

On Linux systems that do not use systemd, Ctrl-Alt-Delete is controlled by the */etc/inittab* file. This example, from MX Linux, shows a typical configuration:

```
# What to do when CTRL-ALT-DEL is pressed.
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
```

12345 makes it active in runlevels 1, 2, 3, 4, and 5. *-t1* means wait one second, *-a* calls */etc/shutdown.allow*, and *-r* is reboot. Configure it to power off the system with the *-h* option, just like running *shutdown* from the command line (see Recipe 3.2). To disable Ctrl-Alt-Delete, comment out the line with the *shutdown* command.

Note that *-t1* and *-a* are not present in all *shutdown* command implementations. The preceding example is from MX Linux. MX Linux supports both the Unix System V initialization system (SysV init) and systemd, and you select the one you want to use from the boot menu.

Ctrl-Alt-Delete is coded into the IBM PC BIOS/UEFI, and it should always reboot a system before the operating system is launched, up to the moment before GRUB launches the operating system.

Ctrl-Alt-Delete was created by IBM engineer David Bradley for the IBM PC BIOS. Orignally it was a developer tool and not intended for users. It required two hands by design, to make it difficult to press by accident.

Then Microsoft adopted it to bring up the Task Manager on the first press, and to reboot on the second press. Then in Windows NT it was used to access the Windows login screen. Supposedly this was a security measure that prevented users from being deceived by fake login screens, which I never knew was a risk, but that was a long time ago. There is a funny exchange about the invention and use of Ctrl-Alt-Delete between Mr. Bradley and Bill Gates preserved in a YouTube video, which hopefully will stay up forever: "Control-Alt-Delete: David Bradley & Bill Gates".

See Also

- *man 7 systemd.special*

3.6 Disabling, Enabling, and Configuring Ctrl-Alt-Delete in the Linux Console

Problem

systemd controls the behavior of Ctrl-Alt-Delete in the Linux console, and you want to know how to manage it.

Solution

You can check the status, disable or enable Ctrl-Alt-Delete, or change it to power off the system.

The Ctrl-Alt-Delete unit file is not a service, but a target, so it does not run as a daemon. If the */etc/systemd/system/ctrl-alt-del.target* symlink exists, Ctrl-Alt-Delete is enabled.

The following example disables and masks *ctrl-alt-del.target*:

```
$ sudo systemctl disable ctrl-alt-del.target
Removed /etc/systemd/system/ctrl-alt-del.target.

$ sudo systemctl mask ctrl-alt-del.target
Created symlink /etc/systemd/system/ctrl-alt-del.target → /dev/null.
```

Unmask and re-enable it:

```
$ sudo systemctl unmask ctrl-alt-del.target
Removed /etc/systemd/system/ctrl-alt-del.target.

$ sudo systemctl enable ctrl-alt-del.target
Created symlink /etc/systemd/system/ctrl-alt-del.target →
/lib/systemd/system/reboot.target.
```

The changes take effect immediately.

Change it to power off the system by linking the *ctrl-alt-del.target* unit to the *poweroff.target* unit. First, disable it to remove the existing symlink, then create the new symlink:

```
$ sudo systemctl disable ctrl-alt-del.target
Removed /etc/systemd/system/ctrl-alt-del.target.

$ sudo ln -s /lib/systemd/system/poweroff.target \
  /etc/systemd/system/ctrl-alt-del.target
```

Now it will power off the system instead of rebooting.

Discussion

Use the *stat* command to see symlinks:

```
$ stat /lib/systemd/system/ctrl-alt-del.target
  File: /lib/systemd/system/ctrl-alt-del.target -> reboot.target
  Size: 13              Blocks: 0          IO Block: 4096   symbolic link
Device: 802h/2050d      Inode: 136890      Links: 1
Access: (0777/lrwxrwxrwx)  Uid: ( 0/ root)   Gid: ( 0/ root)
```

You should not change any */lib/systemd/system/* links. Instead, create the new symlink in */etc/systemd/system/* so that your change is not overwritten by system updates.

See Also

- *man 7 systemd.special*

3.7 Creating Scheduled Shutdowns with cron

Problem

You want your machine to turn itself off at night so you can walk away and not worry about it. Or, your users are careless watt wasters who refuse to develop the habit of shutting their PCs down at night.

Solution

Use *cron* to create scheduled shutdowns. For example, add this line to */etc/crontab* to shut down every night at 10:30 P.M., with a 20-minute warning. Editing */etc/crontab* requires root permissions, and the following example uses the nano text editor:

```
$ sudo nano /etc/crontab
#  m   h   dom mon dow   user    command
  10  22    *   *   *    root    /sbin/shutdown -h +20
```



Before doing this on your work computer, check your employer's policies. If they run updates and backups at night, then your computer may need to stay on.

This example runs only at 11 P.M. on weekdays and shuts down immediately:

```
#  m   h   dom mon dow   user    command
  00  23    *   *  1-5   root    /sbin/shutdown -h now
```

Another way is to use the *crontab* command as root or with *sudo*:

```
$ sudo crontab -e
# m   h     dom mon  dow command
  00  23    *   *    1-5 /sbin/shutdown -h now
```

There is no name field when you run *crontab*, as there is in */etc/crontab*. The previous example opens the root user's crontab in edit mode. Edit and save, then you're done.

Don't try to name the file yourself. During editing it is a temporary file, which is automatically renamed by *crontab* when you save it. It will be saved in */var/spool/cron/crontabs*.

Discussion

/etc/crontab has a name field, so any user can have entries in this file, but only root can edit */etc/crontab*. Users who want to control their own personal crontabs should use the *crontab* command, as personal crontabs do not require root permissions.

The fields in */etc/crontab* can take a little getting used to (Table 3-1), so here are more examples and explanations.

Table 3-1. Table of crontab values

Field	Allowed values
minute	0-59
hour	0-23
day of month	1-31
month	1-12
day of week	0-7

The asterisk * is a wildcard for "all."

Shut down only on weekends:

```
# shutdown at 1:05 am Saturdays and Sundays
00 01   * * 7,0   root /sbin/shutdown -h +5
```

There is a quirk in *cron* that Sunday is 0 or 7. This dates back to very olden times, and I have no idea why it persists. You'll have to test it to see what works, so you may need to use *6,7* for Saturday, Sunday:

```
00 01   * * 6,7   root /sbin/shutdown -h +5
```

It would be nice to use *sat,sun*, but you can only enter one day by name, and cannot use names in lists. Days of the week and months are named with the first three letters: sat, sun, jan, feb. Case does not matter.

You can use ranges: 1–4 means 1, 2, 3, and 4.

Ranges and lists can be mixed: 1, 3, 5, 6-10.

Step values follow ranges:

- 10–23/2 is every second hour in the range
- */2 in the dow field means every other day
- 2–6/2 equals 2, 4, 6

The following strings are nice shortcuts that replace the first five fields:

```
@reboot
@yearly
@annually
@monthly
@weekly
@daily
@midnight
@hourly
```

See Also

- *man 8 cron*
- *man 1 crontab*
- *man 5 crontab*

3.8 Scheduling Automated Startups with UEFI Wake-Ups

Problem

Scheduled shutdowns are so wonderful, you want scheduled wake-ups as well.

Solution

You're in luck, because Linux supports scheduled wake-ups. There are three methods to try: Wake-on-LAN, real-time clock (RTC) wake-ups, or your computer's UEFI setup, if it has a scheduled wake-up feature.

UEFI wake-ups are the most reliable. Figure 3-2 shows the scheduled wake-up screen on a Lenovo ThinkPad.

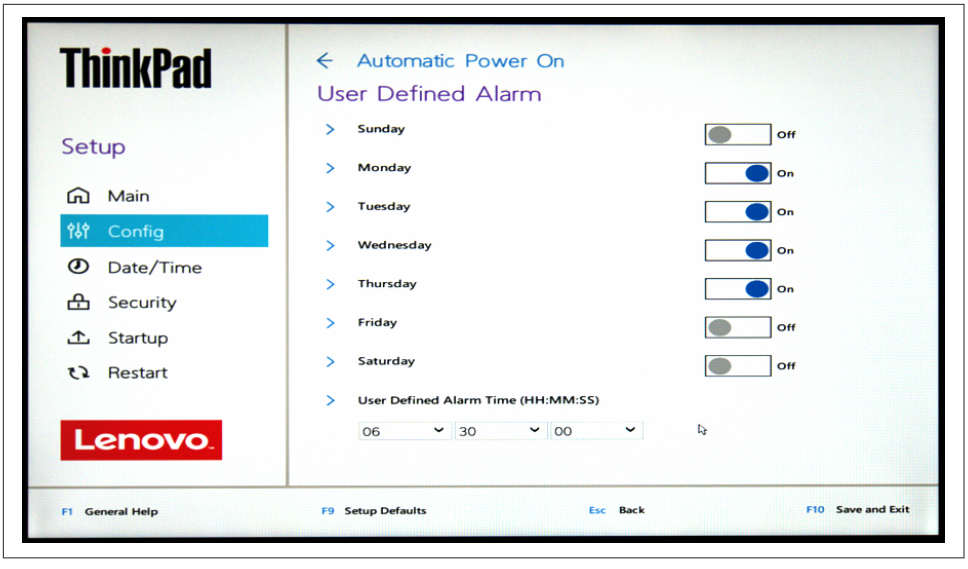


Figure 3-2. Scheduling wake-ups in a Lenovo UEFI

Enter your UEFI setup by pressing the appropriate F*n* key at startup. On Dell, ASUS, and Acer systems this is usually F2; Lenovo uses F1. This varies, however. For example, some systems use the Delete key, so check your machine's documentation. Some systems tell which key to press on their startup screens. It can be a bit tricky to press the key at the right time, so start pressing it right after you press the power button, just like banging on an elevator button to make it arrive faster.

If your system does not have a UEFI wake-up feature, try either Wake-on-LAN (see Recipe 3.10, which requires a second device to send a wake-up signal), or RTC wake-ups (see Recipe 3.9).

Discussion

Let's discuss briefly what BIOS and UEFI mean. When you boot up your computer, the first startup instructions come from the Basic Input Output System (BIOS), or the Unified Extensible Firmware Interface (UEFI) firmware stored on the computer's motherboard. BIOS is the old legacy system that has been with us since 1980. UEFI is its modern replacement. UEFI includes legacy BIOS support, though someday this will be removed. Nearly all computers made after the mid-2000s have UEFI.

UEFI has considerably more features than the old BIOS, and is like a little operating system. The UEFI setup screens control boot order, boot devices, security options, Secure Boot, overclocking, displaying hardware health, networking, and many more functions.

See Also

- Recipe 3.9
- Recipe 3.10
- Recipe 3.11

3.9 Scheduling Automated Startups with RTC Wake-ups

Problem

You want to set up scheduled wake-ups with RTC because your UEFI setup does not have a scheduled wake-up feature, or because you just want to.

Solution

Use the *rtcwake* command, which should already be present on your system from the *util-linux* package. *rtcwake* stops and wakes your system. You may set it to wake up your system after a specified interval, such as 1800 seconds from now, or at a scheduled time and date.

Your system's real-time clock (RTC) should be set to Coordinated Universal Time (UTC).

When *rtcwake* stops your system, it sends it into an ACPI sleep state. Look in */sys/power/state* to see which sleep states your system supports. In the following example, only three of the six ACPI sleep states are supported:

```
$ cat /sys/power/state
freeze mem disk
```

On non-systemd Linuxes, run *cat /proc/acpi/info*.

In the above example, we have three sleep states to try. Test each one according to the following example:

```
$ sudo rtcwake -m freeze -s 60
```

-m specifies the standby mode, and *-s* is the number of seconds until the system starts up again. When it is successful, you will see your system go into a sleep state, then wake up, and you will see either a success message or an error message.

The following example does a dry run of scheduling a wake-up tomorrow at 8 A.M.:

```
$ sudo rtcwake -n -m disk no -u -t $(date +%s -d "tomorrow 08:00")
rtcwake: wakeup from "disk" using /dev/rtc0 at Mon Nov 23 08:00:00 2021
```

Remove *-n* to disable the dry run and run it for real.

The following is a nice, simple */etc/crontab* example to automate shutdowns and wake-ups. The *rtcwake* command suspends to disk at 11 P.M. on weeknights and wakes up 8 hours later:

```
# m   h   dom mon dow   user    command
  00  23    *   *  1-5   root    /usr/sbin/rtcwake -m disk -s 28800
```

Discussion

Look in your system's BIOS/UEFI to verify that your hardware clock is set to UTC. If there is no button or some kind of setting to change the real-time clock to UTC, change the time manually to the current UTC time.

The example that includes *-u -t +$(date +%s -d "tomorrow 08:00")* converts Unix epoch time to human-readable values. Unix epoch time is the number of seconds elapsed since midnight January 1, 1970 UTC. *date +%s* reports the current Unix epoch time. The *-t* option passes Unix epoch time to *date* for conversion, and *-u* specifies that your hardware clock is set to UTC.

The *no* option for *rtcwake* means do not go into a sleep state, but only set the wake-up time. Remove the *no* option to immediately go into a sleep state.

Real-time clock (RTC) wake-ups are the least reliable. Your system has to be put into an Advanced Configuration and Power Interface (ACPI) sleep state supported by your Linux. ACPI is the modern power management standard for managing sleep states. It is supposed to be vendor neutral and hardware independent, but the standard is complex and hardware vendors often support only a subset of its capabilities. Adding to the fun, the various Linuxes implement it in different ways.

There are six ACPI sleep states, S0-S5. The Linux kernel is capable of supporting up to four, and distributions vary in which ones and how many:

S0
: System is running, monitor may be off, most peripherals are on.

S1
: Power-on suspended, CPU stops, power to CPU and RAM is on.

S2
: CPU is powered off, dirty cache is flushed to RAM.

S3
: Also called standby, sleep, and suspend-to-RAM. Data may not be written to disk.

S4
: Hibernation, suspend-to-disk. Everything in RAM is written to disk and the system is powered down.

S5
: Similar to powering the system off, except there is still power to the power button and peripherals, such as the keyboard, network interface, and USB devices.

See Also

- *man 8 cron*
- *man 8 rtcwake*
- Time and Date

3.10 Setting Up Remote Wake-Ups with Wake-on-LAN over Wired Ethernet

Problem

You want to set up remotely triggered wake-ups with Wake-on-LAN because your UEFI setup does not have a scheduled wake-up feature, or it is too simple for your needs, or you want the flexibility to send a wake-up signal at random times. This recipe is for waking up a machine on the same network as the device you use to send the wake-up signal, and your target machine must use a wired Ethernet interface.

Solution

Configure your PC to listen for wake-up requests, then use a second device, such as another computer, a smartphone, or a Raspberry Pi to send the wake-up signal, which is called the *magic packet*. Which isn't really magic, but merely a specialized packet designed specifically for remote wake-ups. (Yes, I too am sad it is not real magic.)

Start by booting into your system's UEFI setup and look for settings for enabling Wake-on-LAN.



An important precaution is to disable all settings that enable PXE (preboot execution environment) booting. If PXE boot is enabled, and you have a PXE server (a preboot execution environment server, which is booting from a network installation server) on your network, it is possible that your machine will wake up to PXE boot and install a new image, overwriting the existing installation.

Then exit and finish starting up. Install the *wakeonlan* and *ethtool* packages.

Fetch the name of your Ethernet interface, which in this example is *enp0s25*, and use *ethtool* to verify that it supports Wake-on-LAN. The output is abbreviated for clarity:

```
$ ip addr show
2: enp0s25: <BROADCAST,MULTICAST,UP,LOWER_UP> state UP 0
    link/ether 9c:ef:d5:fe:8f:20 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.97/24 brd 192.168.1.255 scope global dynamic
    [...]

$ sudo ethtool enp0s25 | grep -i wake-on
        Supports Wake-on: pumbg
        Wake-on: g
```

Record the MAC address of your interface. In the preceding *ip* example, the "ether" line lists the MAC address, 9c:ef:d5:fe:8f:20. Yours will be different, as MAC addresses are unique.

"Supports Wake-on: pumbg" is the magic phrase that verifies your interface has the necessary support, indicated by the *g* switch. The second line, "Wake-on: g," tells you that it is already enabled. If it is not, enable it:

```
$ sudo ethtool -s enp0s25 wol g
```

If it does not stay enabled after a restart, add an entry like this to */etc/crontab* to run the command after every restart:

```
$ @reboot root /usr/bin/ethtool -s enp0s25 wol g
```

Shut down the machine, and from a second device on the same network, send the command to wake it up, using the MAC address of your target machine's Ethernet interface:

```
$ /usr/bin/wakeonlan 9c:ef:d5:fe:8f:20
```

If your target machine and second device are on the same network but in different subnets, specify the broadcast address for your target machine:

```
$ /usr/bin/wakeonlan -i 192.168.44.255 9c:ef:d5:fe:8f:20
```

Discussion

Wake-on-LAN is an Ethernet standard for remotely waking up a computer by sending it a wake-up signal over a network. *wakeonlan* is the name of the command and the name of the package on most Linuxes.

When your computer is shut down, it isn't really turned off, but is in a low power mode, and can receive and act on the magic packet wake-up signal.

wakeonlan sends the magic packet over UDP port 9. The magic packet goes to the network's broadcast address, and every host on the network receives this packet. The MAC address ensures that only the host with that address will wake up.

The target machine wakes up just as if you had pressed the power button.

See Also

- *man 1 wakeonlan*
- *man 8 ethtool*
- Recipe 3.8
- Recipe 3.9
- Recipe 3.11

3.11 Setting Up Remote Wake-Ups over WiFi (WoWLAN)

Problem

Your want to wake up a remote computer via its wireless interface (Wake-on-Wireless LAN, or WoWLAN).

Solution

This recipe is for waking up a machine on the same network as the device you use to send the wake-up signal.

Your machine must have an onboard wireless interface, either integrated on the motherboard or peripheral component interconnect (PCI). It will not work with a USB interface because there is no power to the USB bus when the machine is shut down.

First, enter the UEFI setup of the machine you want to wake up remotely and enable any Wake-on-LAN settings.



An important precaution is to also disable all settings that enable PXE booting. If PXE boot is enabled and you have a PXE server on your network, it is possible that your machine will wake up to PXE boot and install a new image, overwriting the existing installation.

Then exit and finish starting up. Install the *iw* command, and use it to list all of your wireless devices:

```
$ iw dev
phy#0
        Interface wlxcc3fd5fe014c
                ifindex 3
                wdev 0x1
```

```
addr 9c:bf:25:fe:0e:7c
ssid accesspointe
type managed
channel 11 (2462 MHz), width: 20 MHz, center1: 2462 MHz
txpower 20.00 dBm
```

If there is more than one, query the one you want to use. The following example shows a wireless interface that does not support WoWLAN:

```
$ iw phy0 wowlan show
command failed: Operation not supported (-95)
```

The following example is for an interface that supports WoWLAN, and it is not enabled:

```
$ iw phy0 wowlan show
WoWLAN is disabled
```

Enable WoWLAN:

```
$ sudo iw phy0 wowlan enable magic-packet
WoWLAN is enabled:
  * wake up on magic packet
```

iw dev provides the MAC address, which your second device needs to send the magic packet:

```
$ /usr/bin/wakeonlan 9c:bf:25:fe:0e:7c
```

Send the magic packet to a machine on the same network, but in a different subnet, using the broadcast address of the subnet:

```
$ /usr/bin/wakeonlan -i 192.168.44.255 9c:bf:25:fe:0e:7c
```

Discussion

This works well when the remote computer is a laptop, as laptops have integrated wireless network interfaces that typically support more features. Desktop machines usually don't ship with integrated wireless interfaces, so you must shop carefully for a PCI/PCIe wireless adapter that supports WoWLAN and Linux.

See Also

- *man 8 iw*
- *man 1 wakeonlan*
- Recipe 3.8
- Recipe 3.9
- Recipe 3.10

CHAPTER 4

Managing Services with systemd

Every time you start your Linux computer, its initialization system launches a batch of processes, from a few dozen to hundreds, depending on how the system is set up. You can see this on your startup screen (Figure 4-1; press the Escape key to hide your graphical startup screen and see the startup messages).

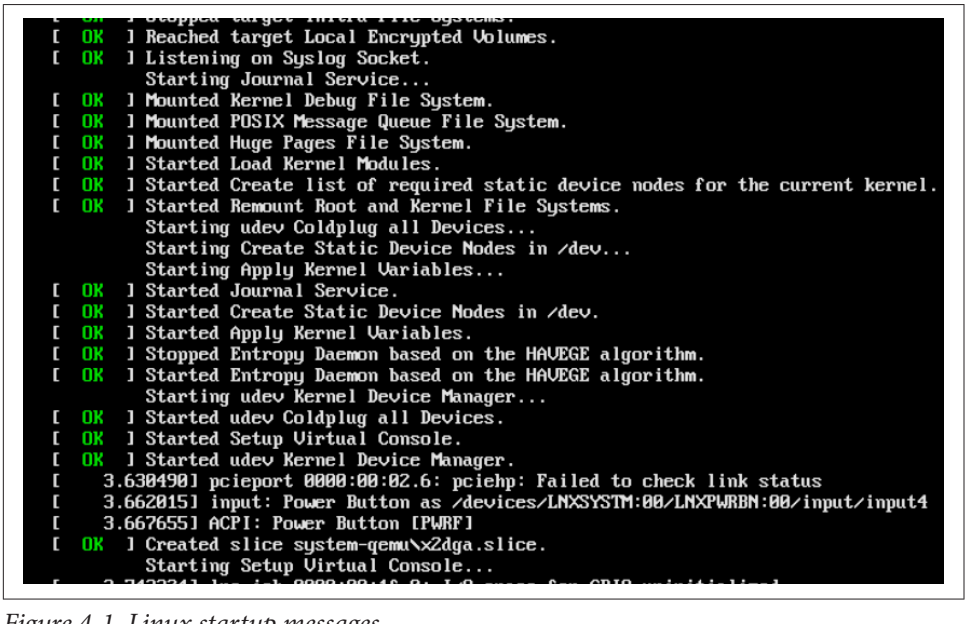


Figure 4-1. Linux startup messages

In olden times we had the Unix System V initialization system (SysV init), BSD init, and Linux Standard Base (LSB) init for launching processes at startup. SysV init was the most common. Those days are fading away, and now systemd is the shiny new

init system for Linux. It has been adopted by all the major Linux distributions, though of course there are a number of distributions that still use the legacy init systems.

In this chapter you will learn if your Linux distribution uses systemd. You will learn what processes, threads, services, and daemons are, and how to use systemd to manage services: start, stop, enable, disable, and check status. You will become acquainted with the *systemctl* command, which is the systemd system and service manager.

systemd is designed to provide functionality suited to modern complex server and desktop systems, and it does considerably more than the legacy init systems. It provides complete service management from startup to shutdown, starting processes at boot, on-demand after boot, and shutting down services when they are not needed. It manages functions such as system logging, automounting filesystems, automatic service dependency resolution, name services, device management, network connection management, login management, and a host of other tasks.

This sounds like a lot until you realize that processes do everything on a computer, and all of this functionality used to be provided by a large assortment of other programs. systemd brings it all together in an integrated software suite that should operate the same way on all Linux systems, though as always with Linux there are some minor exceptions, such as file locations and service names. Be aware that your particular Linux may have some differences from the examples in this chapter.

systemd attempts to decrease boot times and parcel out system resources more efficiently by starting processes concurrently and in parallel, and starting only necessary services, leaving other services to start after boot as needed. A service that is dependent on other services no longer has to wait to start for those services to become available because all it needs is a waiting Unix socket to become available. Recipe 4.9 shows how to find processes that are slowing down your system startup.

systemd binaries are written in C, which provides some performance enhancement. The legacy inits are masses of shell scripts, and any compiled language operates faster than shell scripts.

systemd is backwards compatible with SysV init. Most Linux distributions retain the legacy SysV configuration files and scripts, including */etc/inittab* and the */etc/rc.d/* and */etc/init.d/* directories. When a service does not have a systemd configuration file, systemd looks for a SysV configuration file. systemd is also backward compatible with Linux Standard Base (LSB) init.

systemd service files are smaller and easier to understand than SysV init files. Compare a SysV init file for sshd with its systemd service file. This is a snippet of the sshd init file, */etc/init.d/ssh*, from MX Linux:

```
#! /bin/sh

### BEGIN INIT INFO
# Provides:             sshd
# Required-Start:       $remote_fs $syslog
# Required-Stop:        $remote_fs $syslog
# Default-Start:        2 3 4 5
# Default-Stop:
# Short-Description:    OpenBSD Secure Shell server
### END INIT INFO

set -e

# /etc/init.d/ssh: start and stop the OpenBSD "secure shell(tm)" daemon

test -x /usr/sbin/sshd || exit 0

umask 022

if test -f /etc/default/ssh; then
[...]
```

This goes on for a total of 162 lines. This is a complete systemd service file from Ubuntu 20.04, */lib/systemd/system/ssh.service*:

```
[Unit]
Description=OpenBSD Secure Shell server
Documentation=man:sshd(8) man:sshd_config(5)
After=network.target auditd.service
ConditionPathExists=!/etc/ssh/sshd_not_to_be_run

[Service]
EnvironmentFile=-/etc/default/ssh
ExecStartPre=/usr/sbin/sshd -t
ExecStart=/usr/sbin/sshd -D $SSHD_OPTS
ExecReload=/usr/sbin/sshd -t
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartPreventExitStatus=255
Type=notify
RuntimeDirectory=sshd
RuntimeDirectoryMode=0755

[Install]
WantedBy=multi-user.target
Alias=sshd.service
```

Even without reading the documentation, or knowing anything about systemd, you can understand some of what this file is supposed to do.

See Rethinking PID 1 for a detailed introduction to systemd by one of its inventors and maintainers, Lennart Poettering. Rethinking PID 1 details the rationale behind building a new init system, its architecture, advantages, and how it uses existing Linux kernel features in place of duplicating existing functionality.

4.1 Learning if Your Linux Uses systemd

Problem

You need to know if your Linux distribution uses systemd or something else.

Solution

Look for the */run/systemd/system/* directory. If this exists, then your init system is systemd.

Discussion

The */run/systemd/* directory may be present on your system if your distribution supports multiple init systems. But systemd is not the active init unless you see */run/systemd/system/*.

There are several other ways to learn which init system your system is using. Try querying */sbin/init*. Originally this was the SysV executable, and now most Linux distributions preserve the name and symlink it to the systemd executable. This example confirms that the init is systemd:

```
$ stat /sbin/init
File: /sbin/init -> /lib/systemd/systemd
[...]
```

On a system using SysV init, it has no symlink:

```
$ stat /sbin/init
File: /sbin/init
[...]
```

The */proc* pseudofilesystem is an interface to your Linux kernel, and contains the current state of a running system. It is called a pseudofilesystem because it exists only in memory and not on disk. In this example, */proc/1/exe* is symlinked to the systemd executable:

```
$ sudo stat /proc/1/exe
File: /proc/1/exe -> /lib/systemd/systemd
[...]
```

On a SysV system, it links to *init*:

```
$ sudo stat /proc/1/exe
File: /proc/1/exe -> /sbin/init
[...]
```

The */proc/1/comm* file reports your active init system:

```
$ cat /proc/1/comm
systemd
```

On a SysV system, it reports *init*:

```
$ cat /proc/1/comm
init
```

The command attached to process ID (PID) 1 is your init. PID 1 is the first process launched at startup, which then starts all other processes. You can see this with the *ps* command:

```
$ ps -p 1
  PID TTY          TIME CMD
    1 ?        00:00:00 systemd
```

When the init is SysV, it looks like this:

```
$ ps -p 1
  PID TTY          TIME CMD
    1 ?        00:00:00 init
```

See Recipe 4.2 for more information on PID 1.

Linux support for systemd varies. Most of the major Linux distributions have adoped systemd, including Fedora, Red Hat, CentOS, openSUSE, SUSE Linux Enterprise, Debian, Ubuntu, Linux Mint, Arch, Manjaro, Elementary, and Mageia Linux.

Some popular distributions that do not support systemd, or include it but not as the default init, are Slackware, PCLinuxOS, Gentoo Linux, MX Linux, and antiX.

See Also

- Distrowatch for information on hundreds of Linux distributions
- *man 5 proc*
- *man 1 pstree*
- *man 1 ps*

4.2 Understanding PID 1, the Mother of All Processes

Problem

You want a better understanding of services and processes on Linux.

Solution

PID 1 is the mother of all processes on Linux systems. This is the first process to start, and then it launches all other processes.

Processes are one or more running instances of a program. Every task in a Linux system is performed by a process. Processes can create independent copies of themselves, that is, they can *fork*. The forked copies are called *children*, and the original is the *parent*. Each child has its own unique PID, and its own allocation of system resources, such as CPU and memory. *Threads* are lightweight processes that run in parallel and share system resources with their parents.

Some processes run in the background and do not interact with users. Linux calls these processes *services* or *daemons*, and their names tend to end with the letter D, such as httpd, sshd, and systemd.

Every Linux system starts PID 1 first, which then launches all other processes. Use the *ps* command to list all running processes in PID order:

```
$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 10:06 ?        00:00:01 /sbin/init splash
root         2     0  0 10:06 ?        00:00:00 [kthreadd]
root         3     2  0 10:06 ?        00:00:00 [rcu_gp]
root         4     2  0 10:06 ?        00:00:00 [rcu_par_gp]
[...]
```

The *pstree* command organizes this mass of information into a tree diagram. This example shows all processes, their child processes, PIDs, and threads, which are enclosed in curly braces:

```
$ pstree -p
systemd(1)─┬─ModemManager(925)─┬─{ModemManager}(944)
           │                   └─{ModemManager}(949)
           ├─NetworkManager(950)─┬─dhclient(1981)
           │                     ├─{NetworkManager}(989)
           │                     └─{NetworkManager}(991)
           ├─accounts-daemon(927)─┬─{accounts-daemon}(938)
           │                      └─{accounts-daemon}(948)
           ├─acpid(934)
           ├─agetty(1103)
           ├─avahi-daemon(953)───avahi-daemon(970)
[...]
```

The full *pstree* output is quite large. You can view a single process, identified by its PID, and its parents, children, and threads, like the following example for the Kate text editor:

```
$ pstree -sp 5193
systemd(1)───kate(5193)─┬─bash(5218)
                        ├─{kate}(5195)
                        ├─{kate}(5196)
                        ├─{kate}(5197)
                        ├─{kate}(5198)
                        ├─{kate}(5199)
[...]
```

This shows that `systemd(1)` is Kate's parent, `bash(5218)` is Kate's child, and all the processes in curly braces are Kate's threads.

Discussion

Processes always exist in one of several states, and these states change according to system activity. The following *pstree* example displays the PID, user, state, and command fields:

```
$ ps -eo pid,user,stat,comm
  PID USER       STAT COMMAND
    1 root       Ss   systemd
    2 root       S    kthreadd
   32 root       I<   kworker/3:0H-kb
   68 root       SN   khugepaged
11222 duchess    Rl   konsole
```

- *R* is either currently running or waiting in the run queue.
- *l* means the process is multithreaded.
- *S* is interruptable sleep; the process is waiting for an event to complete.
- *s* is a session leader. Sessions are related processes managed as a unit.
- *I* is an idle kernel thread.
- < means high priority.
- *N* is low priority.

There are several rarely used states you can read about in *man 1 ps*.

See Also

- Recipe 4.7
- *man 5 proc*

- *man 1 pstree*
- *man 1 ps*

4.3 Listing Services and Their States with systemctl

Problem

You want to list all services installed on your system, and you want to know the states of the services: whether they are running, not running, or in an error state.

Solution

systemctl, the systemd manager command, tells all. Run it with no options to see a detailed list of all loaded units. A systemd unit is any related batch of processes defined in a unit configuration file and managed by systemd:

```
$ systemctl
```

This prints a giant pile of information: 177 active loaded units on my test system with the full unit names, status, and long descriptions. Redirect the the output to a text file for easier study:

```
$ systemctl > /tmp/systemctl-units.txt
```

Treat yourself to more information overload by listing all units, active and inactive:

```
$ systemctl --all
```

This results in 349 loaded units listed on my test system, including *not-found* and *inactive* units. How many total unit files? The following example shows 5 out of 322:

```
$ systemctl list-unit-files
UNIT FILE                                   STATE
proc-sys-fs-binfmt_misc.automount           static
-.mount                                     generated
mount                                       generated
dev-hugepages.mount                         static
home.mount                                  generated
[...]
322 unit files listed.
```

We are interested in service files because Linux users and administrators interact mainly with service files and rarely need to bother with any other type of unit file. How many are installed? Let's see:

```
$ systemctl list-unit-files --type=service
UNIT FILE                               STATE
accounts-daemon.service                 enabled
acpid.service                           disabled
```

```
alsa-state.service                          static
alsa-utils.service                          masked
anacron.service                             enabled
[...]
212 unit files listed.
```

The preceding example displays the four most common states that a service can be in: enabled, disabled, static, or masked.

List only enabled services:

```
$ systemctl list-unit-files --type=service --state=enabled
UNIT FILE                                   STATE
accounts-daemon.service                     enabled
anacron.service                             enabled
apparmor.service                            enabled
autovt@.service                             enabled
avahi-daemon.service                        enabled
[...]
62 unit files listed.
```

List only disabled services:

```
$ systemctl list-unit-files --type=service --state=disabled
UNIT FILE                               STATE
acpid.service                           disabled
brltty.service                          disabled
console-getty.service                   disabled
mariadb@.service                        disabled
[...]
12 unit files listed.
```

List only static services:

```
$ systemctl list-unit-files --type=service --state=static
UNIT FILE                                 STATE
alsa-restore.service                      static
alsa-state.service                        static
apt-daily-upgrade.service                 static
apt-daily.service                         static
[...]
106 unit files listed.
```

List only masked services:

```
$ systemctl list-unit-files --type=service --state=masked
UNIT FILE                     STATE
alsa-utils.service            masked
bootlogd.service              masked
bootlogs.service              masked
checkfs.service               masked
[...]
36 unit files listed.
```

Discussion

Service unit files are in */usr/lib/systemd/system/* or */lib/systemd/system/*, according to where your Linux distribution puts them. These are plain-text files you can read.

enabled
: This shows that the service has become available and is managed by systemd. When a service is enabled, systemd creates a symlink in */etc/systemd/system/* from the unit file in */lib/systemd/system/*. It can be started, stopped, reloaded, and disabled by the user with the *systemctl* command.



Enabling a service does not immediately start it, and disabling a service does not immediately stop it (see Recipe 4.6).

disabled
: Diabled means that there is no symlink in */etc/systemd/system/*, and it will not start automatically at boot. You can stop and start it manually.

masked
: This means the service is linked to */dev/null/*. It is completely disabled and cannot be started by any means.

static
: This means that the unit file is a dependency of other unit files, and cannot be started or stopped by the user.

Some less-common service states you will see:

indirect
: Indirect states belong to services that are not meant to be managed by users, but are meant to be used by other services.

generated
: Generated states indicate that the service has been converted from a nonnative systemd initialization configuration file, either SysV or LSB init.

See Also

- *man 1 systemctl*

4.4 Querying the Status of Selected Services

Problem

You want to know the status of one service or a few specific services.

Solution

systemctl status provides a nice little bundle of useful status information. The following example queries the CUPS service. CUPS, the Common Unix Printing System, should be on all Linux systems:

```
$ systemctl status cups.service
● cups.service - CUPS Scheduler
     Loaded: loaded (/lib/systemd/system/cups.service; enabled; vendor preset:
             enabled)
     Active: active (running) since Sun 2021-11-22 11:01:48 PST; 4h 17min ago
TriggeredBy: ● cups.path
             ● cups.socket
       Docs: man:cupsd(8)
   Main PID: 1403 (cupsd)
      Tasks: 2 (limit: 18760)
     Memory: 3.8M
     CGroup: /system.slice/cups.service
             ├─1403 /usr/sbin/cupsd -l
             └─1421 /usr/lib/cups/notifier/dbus dbus://

Nov 22 11:01:48 host1 systemd[1]: Started CUPS Scheduler.
```

Query multiple services with a space-delimited list:

```
$ systemctl status mariadb.service bluetooth.service lm-sensors.service
```

Discussion

There is a lot of useful information in this little bit of output (Figure 4-2).

```
duchess@client4:~$ systemctl status cups.service
● cups.service - CUPS Scheduler
     Loaded: loaded (/lib/systemd/system/cups.service; enabled; vendor p>
     Active: active (running) since Sun 2021-04-18 10:21:28 PDT; 1h 30mi>
TriggeredBy: ● cups.path
             ● cups.socket
       Docs: man:cupsd(8)
   Main PID: 991 (cupsd)
      Tasks: 1 (limit: 18755)
     Memory: 2.6M
     CGroup: /system.slice/cups.service
             └─991 /usr/sbin/cupsd -l
```

Figure 4-2. systemctl status output for the CUPS printer service

The dot next to the service name is a quick status indicator. It appears in colors on most terminals. White is an *inactive* or *deactivating* state. Red is a *failed* or *error* state. Green indicates an *active*, *reloading*, or *activating* state. The rest of the information in the output is described in the following:

Loaded
: Verifies that the unit file has been loaded into memory, displays its full path, the service is enabled (see the Discussion about states in Recipe 4.3), and `vendor preset: disabled/enabled` indicates if the installation default is to start at boot or not. When it is disabled, the vendor default is to not start at boot. This only shows the vendor preference and does not indicate if it is currently enabled or disabled.

Active
: Tells you if the service is active or inactive, and how long it has been in that state.

Process
: Reports the PIDs and their commands and daemons.

Main PID
: This is the process number for the cgroup slice.

Tasks
: Reports how many tasks the service has started. Tasks are PIDs.

CGroup
: Shows which unit slice the service belongs to and its PID. The three default unit slices are *user.slice*, *system.slice*, and *machine.slice*.

  Linux control groups (cgroups) are sets of related processes and all of their future children. In systemd, a *slice* is a subdivision of a cgroup, and each slice manages a particular group of processes. Run *systemctl status* to see a diagram of the cgroup hierarchy.

  By default, service and scope units are grouped in */lib/systemd/system/system.slice*.

  User sessions are grouped in */lib/systemd/system/user.slice*.

  Virtual machines and containers registered with systemd are grouped in */lib/systemd/system/machine.slice*.

The remaining lines are the most recent log entries from *journalctl*, the systemd log manager.

See Also

- *man 1 systemctl*
- *man 5 systemd.slice*
- *man 1 journalctl*
- Kernel cgroups documentation

4.5 Starting and Stopping Services

Problem

You want to stop and start services with systemd.

Solution

This is a job for *systemctl*. The following examples use the SSH service to demonstrate service management.

Start a service:

```
$ sudo systemctl start sshd.service
```

Stop a service:

```
$ sudo systemctl stop sshd.service
```

Stop and then restart a service:

```
$ sudo systemctl restart sshd.service
```

Reload the service's configuration. For example, you made a change to *sshd_config* and want to load the new configuration without restarting the service:

```
$ sudo systemctl reload sshd.service
```

Discussion

All of these commands also work with multiple services, space-delimited, for example:

```
$ sudo systemctl start sshd.service mariadb.service firewalld.service
```

If you're curious about the commands that systemd runs behind the scenes to start, reload, or stop the individual daemons, look in their unit files. Some services have start, reload, stop, and other instructions in their unit files, like this example for httpd:

```
ExecStart=/usr/sbin/httpd/ $OPTIONS -DFOREGROUND
ExecReload=/usr/sbin/httpd $OPTIONS -k graceful
ExecStop=/bin/kill -WINCH ${MAINPID}
```

You don't have to do anything special with this information; it is there when you want to know how *systemctl* is managing a particular service.

See Also

- Recipe 4.6
- *man 1 systemctl*

4.6 Enabling and Disabling Services

Problem

You want a service or services to automatically start at boot, or you want to prevent a service from starting at boot, or to disable it completely.

Solution

Enabling a service configures it to automatically start at boot.

Disabling a service stops it from starting at boot, but it can be started and stopped manually.

Masking a service disables it so that it cannot be started at all.

The following example enables the *sshd* service:

```
$ sudo systemctl enable sshd.service
Created symlink /etc/systemd/system/multi-user.target.wants/sshd.service →
/usr/lib/systemd/system/sshd.service
```

The output shows that enabling a service means creating a symlink from the service file in */lib/systemd/system/* to */etc/systemd/system/*. This does not start the service. You can start the service with *systemctl start*, or enable and start the service in one command with the *--now* option:

```
$ sudo systemctl enable --now sshd.service
```

This command disables the *sshd* service. It does not stop the service, so you must stop it manually after disabling it:

```
$ sudo systemctl disable sshd.service
Removed /etc/systemd/system/multi-user.target.wants/sshd.service
$ sudo systemctl stop sshd.service
```

Or, disable and stop it with one command:

```
$ sudo systemctl disable --now sshd.service
```

This command reenables the *mariadb* service, which disables and then enables it. If you have created the symlinks manually for a service, this is useful for quickly resetting them to the defaults:

```
$ sudo systemctl reenable mariadb.service
Removed /etc/systemd/system/multi-user.target.wants/mariadb.service.
Removed /etc/systemd/system/mysqld.service.
Removed /etc/systemd/system/mysql.service.
Created symlink /etc/systemd/system/mysql.service →
/lib/systemd/system/mariadb.service.
Created symlink /etc/systemd/system/mysqld.service →
/lib/systemd/system/mariadb.service.
Created symlink /etc/systemd/system/multi-user.target.wants/mariadb.service →
/lib/systemd/system/mariadb.service.
```

The following command disables the *bluetooth* service completely by masking it, so that it cannot be started at all:

```
$ sudo systemctl mask bluetooth.service
Created symlink /etc/systemd/system/bluetooth.service → /dev/null.
```

Unmasking the *bluetooth* service does not enable it, so it must be started manually:

```
$ sudo systemctl unmask bluetooth.service
Removed /etc/systemd/system/bluetooth.service.
$ sudo systemctl start bluetooth.service
```

Discussion

When you enable, disable, mask, or unmask a service, it remains in its current state unless you use the *--now* option. The *--now* option works with *enable*, *disable*, and *mask* to immediately stop or start the service, but it does not work with *unmask*.

See the Discussion in Recipe 4.3 to learn more about how systemd uses symlinks to manage services.

See Also

- *man 1 systemctl*
- The Discussion in Recipe 4.3 to learn how systemd uses symlinks to manage services

4.7 Stopping Troublesome Processes

Problem

You want to know how to stop troublesome processes. A certain service may be unresponsive or running away, spawning forks and causing your system to hang. Your normal stop command is not working. What do you do?

Solution

Stopping a process is called killing the process. On Linux systems with systemd, you should use *systemctl kill*. On systems without systemd, use the legacy *kill* command.

systemctl kill is preferable because it stops all processes that belong to a service and leaves no orphan processes, nor any processes that might restart the service and continue to make trouble. First, try it with no options other than the service name, then check the status:

```
$ sudo systemctl kill mariadb

$ systemctl status mariadb
● mariadb.service - MariaDB 10.1.44 database server
   Loaded: loaded (/lib/systemd/system/mariadb.service; enabled; vendor preset:
enabled)
   Active: inactive (dead) since Sun 2020-06-28 19:57:49 PDT; 6s ago
[...]
```

The service has cleanly stopped. If this does not work, then try the nuclear option:

```
$ sudo systemctl kill -9 mariadb
```

The legacy *kill* command does not recognize service or command names, but rather requires the PID of the offending process:

```
$ sudo kill 1234
```

If this does not stop it, use the nuclear option:

```
$ sudo kill -9 1234
```

Discussion

Use the *top* command to identify runaway processes. Run it with no options, and the processes using up the most CPU resources are listed at the top. Press the q key to exit *top*.

```
$ top
top - 20:30:13 up  4:24,  6 users,  load average: 0.00, 0.03, 0.06
Tasks: 246 total,   1 running, 170 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.4 us,  0.2 sy,  0.0 ni, 99.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 16071016 total,  7295284 free,  1911276 used,  6864456 buff/cache
```

```
KiB Swap:  8928604 total,  8928604 free,        0 used. 13505600 avail Mem

  PID USER       PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 3504 madmax     20   0 99.844g 177588  88712 S   2.6  1.1   0:08.68 evolution
 2081 madmax     20   0 3818636 517756 177744 S   0.7  3.2   5:07.56 firefox
 1064 root       20   0  567244 148432 125572 S   0.3  0.9  12:54.75 Xorg
 2362 stash      20   0 2997732 230508 145444 S   0.3  1.4   0:40.72 Web Content
[...]
```

kill sends signals to processes, and the default signal is SIGTERM (signal terminate). SIGTERM is gentle, allowing processes to shut down cleanly. SIGTERM is also ignorable, and processes don't have to pay attention to it. Signals can be identified by name or number; for most folks the numbers are easier to remember, so spelling out the default looks like this:

```
$ sudo kill -1 1234
```

kill -9 is SIGKILL. SIGKILL stops processes immediately and uncleanly, and also attempts to stop all child processes.

Killing services with *systemctl kill* is easier than with *kill*, and more reliable. You only need the service name, and you don't have to hunt down PIDs. It ensures that all processes belonging to the service are stopped, which *kill* cannot ensure.

There are a ton of signals that have accumulated over the years, and you can read all about them in *man 7 signal*. In my experience, the most relevant signals are SIGTERM and SIGKILL, but don't let that stop you from learning more about the others.

If you are uncomfortable with terminology like kill, parents, children, and orphans, so am I. Maybe someday it will change.

See Also

- *man 5 systemd.kill*
- *man 1 systemctl*
- *man 1 kill*
- *man 7 signal*

4.8 Managing Runlevels with systemd

Problem

You want to reboot to different system states in a manner similar to using SysV runlevels.

Solution

systemd *targets* are similar to SysV runlevels. These are boot profiles that start your system with different options, such as multiuser mode with a graphical desktop, multiuser mode with no graphical desktop, and emergency and rescue modes to use when your current target will not boot. (See the Discussion for more information on runlevels.)

The following command checks if the system is running and reports its state:

```
$ systemctl is-system-running
running
```

What is the default target?

```
$ systemctl get-default
graphical.target
```

Get the current runlevel:

```
$ runlevel
N 5
```

Reboot to rescue mode:

```
$ sudo systemctl rescue
```

Reboot to emergency mode:

```
$ sudo systemctl emergency
```

Reboot to the default mode:

```
$ sudo systemctl reboot
```

Reboot to a different target without changing the default:

```
$ sudo systemctl isolate multi-user.target
```

Set a different default runlevel:

```
$ sudo systemctl set-default multi-user.target
```

List the runlevel target files and their symlinks on your system:

```
$ ls -l /lib/systemd/system/runlevel*
```

List the dependencies in a runlevel target:

```
$ systemctl list-dependencies graphical.target
```

Discussion

SysV runlevels are different states that your system can boot to, for example, with a graphical desktop, without a graphical desktop, and with emergency runlevels to use when your default runlevel has problems and will not boot.

systemd *targets* approximately correspond to the legacy SysV runlevels:

- *runlevel0.target*, *poweroff.target*, halt
- *runlevel1.target*, *rescue.target*, single-user text mode, all local filesystems mounted, root user only, no networking
- *runlevel3.target*, *multi-user.target*, multiuser text mode (no graphical environment)
- *runlevel5.target*, *graphical.target*, multiuser graphical mode
- *runlevel6.target*, *reboot.target*, reboot

systemctl emergency is a special target that is more restricted than *rescue* mode: no services, no mount points other than the root filesystem, no networking, root user only. It is the most minimal running system for debugging problems. You may see options to boot into a rescue or emergency mode in your GRUB2 bootloader screen.

systemctl is-system-running reports various system states:

- *initializing* means the system has not completed startup.
- *starting* means the system is in the final stages of startup.
- *running* is fully operational, and all processes are started.
- *degraded* means the system is operational, but one or more systemd units have failed. Run *systemctl | grep failed* to see which units failed.
- *maintenance* means that either the *rescue* or *emergency* target is active.
- *stopping* means that systemd is shutting down.
- *offline* means that systemd is not running.
- *unknown* means that there is a problem preventing systemd from determining the operational state.

See Also

- *man 1 systemctl*
- *man 8 systemd-halt.service*

4.9 Diagnosing Slow Startups

Problem

systemd promises faster startups, but your system starts up slowly, and you want to find out why.

Solution

You want *systemd-analyze blame*. Run it with no options to see a list of system processes and how long they took to start:

```
$ systemd-analyze blame
         34.590s apt-daily.service
          6.782s NetworkManager-wait-online.service
          6.181s dev-sda2.device
          4.444s systemd-journal-flush.service
          3.609s udisks2.service
          2.450s snapd.service
          [...]
```

Analyze only user processes:

```
$ systemd-analyze blame --user
          3.991s pulseaudio.service
           553ms at-spi-dbus-bus.service
           380ms evolution-calendar-factory.service
           331ms evolution-addressbook-factory.service
           280ms xfce4-notifyd.service
           [...]
```

Discussion

It is useful to review everything that starts at boot and perhaps find services you don't want starting at boot. My favorite to disable is Bluetooth because I don't use it on my servers or PCs, but many Linux distros enable it by default.

See Also

- *man 1 systemd-analyze*

CHAPTER 5

Managing Users and Groups

Linux has two types of users: human users and system users. Each user has a unique identity (UID), and at least one group identification (GID). All users have one primary group and may be members of multiple groups.

Each human user owns a home directory for their personal files. User home directories belong in */home* and are named for the owner, like our example user Duchess, who owns */home/duchess*. Users may belong to multiple groups, and the additional group memberships are called *supplemental* groups. Users in a group have all the privileges of that group. (To learn all about privileges, see Chapter 6.) Privileges control access to files and commands, and are fundamental to system security.

System users represent system services and processes. System users need user accounts for controlling their privileges and do not have logins or directories in */home*.

Human users are divided into two categories: the *root* user, or the superuser, is all-powerful and can do anything on the system. All other users are called normal or unprivileged users. Normal users are given just enough privileges to manage their own files and run commands that allow normal users to use them. Normal users can be given limited or complete root powers, which you will learn about in the recipes about *su* and *sudo*.

You can see all the users on your system in */etc/passwd*, and all the groups in */etc/group*.



Centralized User Management

/etc/passwd and */etc/group* are inherited from Unix and have not changed much since they were ported to Linux in 1992. Since then, newer tools have evolved to manage users and groups, such as centralized databases that serve entire organizations. This chapter does not cover centralized user management tools.

Linux comes with a number of commands for managing users and groups:

- *useradd* creates new users.
- *groupadd* creates new groups.
- *userdel* deletes users.
- *groupdel* deletes groups.
- *usermod* is for making changes to existing users.
- *passwd* creates and changes passwords.

These are part of the *Shadow Password Suite*, and */etc/login.defs* is its main configuration file.

useradd behaves differently on different systems, according to how it is configured. Traditionally, it lumped all new users into the same primary group, *users* (100). This meant that users had to be careful with permissions on their files to avoid exposing them to other group users. Red Hat changed this with its *User Private Group* scheme, which creates a personal private group for each new user. Most Linux distributions make this the default, though there are exceptions, such as openSUSE.

The Shadow Password Suite was created by Julianne Frances Haugh way back in the 1980s, back before Linux was born, to improve Unix password security and to make user account management easier. It was ported to Linux in 1992, when Linux was barely a year old.

Before the Shadow Password Suite, all the relevant files had to be edited individually, there were multiple password management commands, and hashed passwords were stored in */etc/passwd* and */etc/group*. These two files must be world readable, so storing passwords in them, even when they're hashed, is asking for trouble. Anyone can copy a world-readable file, and then crack the passwords at their leisure. Relocating the hashed passwords to the shadow files, */etc/shadow* and */etc/gshadow*, which are accessible only by root, added a strong layer of protection. The longevity of the Shadow Password Suite is a testament to how well it was designed and coded.

Newer arrivals are *adduser* and *addgroup* for Debian. They are Perl script wrappers for *useradd* and *groupadd*. These scripts walk you through a complete new user and new group configuration.

In this chapter, you will learn how to create and remove human and system users, manage passwords, find UIDs and GIDs, set your desired defaults for creating new users, change group memberships, customize the common files your new users need, clean up after users that you have removed, become root, and grant limited root powers to normal users.

5.1 Finding a User's UID and GID

Problem

You want to list users' UIDs and GIDs.

Solution

Use the *id* command with no options to see your own UID and GIDs. In the following example, the user is Duchess:

```
duchess@pc:~$ id
uid=1000(duchess) gid=1000(duchess)
groups=1000(duchess),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),118(lpadmin),
126(sambashare),131(libvirt)
```

Display another user's UID and GIDs by providing their username as an argument:

```
duchess@pc:~$ id madmax
uid=1001(madmax) gid=1001(madmax) groups=1001(madmax),1010(composers)
```

Display your effective ID. This is your ID when you run a command as another user. You can see this with *sudo*:

```
duchess@client4:~$ sudo id -un
root

duchess@client4:~$ sudo -u madmax id -gn
madmax
```

Discussion

There are three types of user IDs in Linux:

- Real UID/GID
- Effective UID/GID
- Saved UID/GID

The *real ID* is the UID and primary GID assigned to the user at creation. These are what you see when you run the *id* command, as yourself, with no options.

The *effective ID* is the UID used to run a process that requires different privileges than the user who launched the process, for example, the *passwd* command. *passwd* requires root privileges but uses the special permission modes to allow users to change their own passwords.

You can see this for yourself. First, take a look at the *passwd* command's permissions:

```
$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 68208 May 27  2020 /usr/bin/passwd
```

This shows that *passwd* is owned by root, both UID and GID. Now type the *passwd* command and press Enter.

Open a second terminal to find the process for *passwd*, then print its process ID, effective ID, and real ID:

```
$ ps -a|grep passwd
12916 pts/1    00:00:00 passwd

$ ps -eo pid,euser,ruser,rgroup | grep 12916
  12916 root     root     root
```

Even though an unprivileged user is running *passwd*, it runs with root permissions. (See Recipe 6.11 for information on the special permission modes.)

The *saved ID* is used by processes that need elevated privileges, usually root privileges. When a process needs to do work that requires fewer privileges, it can temporarily switch to a nonprivileged user ID. The effective UID is changed to the lower privilege value, and the original effective UID is saved to the SUID, saved user ID. When the process needs elevated privileges again, it changes to the SUID.

The *id* command has a few options:

- *-u* shows the effective UID number.
- *-g* shows the effective GID number.
- *-G* shows all group IDs.
- *-n* prints the name rather than the number. You can use this in combination with *-u*, *-g*, and *-G*.
- *-un* shows the effective UID username.
- *-gn* shows the effective group name.
- *-Gn* shows all effective GID names.
- *-r* shows the real ID instead of the effective ID. You can use this in combination with *-u*, *-g*, and *-G*.

See Also

- Recipe 6.11
- *man 1 id*
- *man 1 ps*

5.2 Creating a Human User with useradd

Problem

You want to create a new user with a user private group and home directory populated with a set of default files like *.bashrc*, *.profile*, *.bash_history*, and any other files you want them to have.

Solution

The *useradd* command is included in most Linux distributions and is configurable to suit your requirements. The default configuration varies across the various Linux distributions, so the quickest way to learn how your system is set up is to create a new test user:

```
$ sudo useradd test1
```

Now run the *id* command, and then see if *useradd* created a home directory. The following examples are from Fedora 34:

```
$ id test1
uid=1011(test1) gid=1011(test1) groups=1011(test1)

$ sudo ls -a /home/test1/
.  ..  .bash_logout  .bash_profile  .bashrc
```

In this example, the default configuration meets all the requirements listed in the Problem. Now you only need to set a password:

```
$ sudo passwd test1
Changing password for user test1.
New password: password
Retype new password: password
passwd: all authentication tokens updated successfully.
```

You may elect to force the user to reset their password at first login, after creating the user's password:

```
$ sudo passwd -e test1
Expiring password for user test1.
passwd: Success
```

Give the login to your user, and they can start using their new account. The new user account is represented like this in */etc/passwd*:

```
test1:x:1011:1011::/home/test1:/bin/bash
```

Some Linuxes, for example openSUSE, configure *useradd* to not create the user's home directory by default and to put all users into the *users (100)* group. This potentially exposes files to other users, if group permissions on the files allow it. The following example creates a user private group:

```
$ sudo useradd -mU test2
```

-m creates the user's home directory, and *-U* creates their private group with the same name as their username.

Discussion

All new user accounts are inactive until you set a password.

The first group created for a user, whether it is a user private group or a common group for all users, is their *primary* group. All other groups the user is assigned to are *supplementary* groups.

There are some additional useful options:

- *-G*, *--groups* is for adding the user to multiple supplemental groups in a comma-delimited list. The groups must already exist:

  ```
  $ sudo useradd -G group1,group2,group3 test1
  ```

- *-c*, *--comment* accepts any text string. Use this for the user's full name, or any comment or description:

  ```
  $ useradd -G group1,group2,group3 -c 'Test 1,,,,' test1
  ```

The four commas define five fields: full name, room number, work phone, home phone, and other. Way back in olden times this was called the GECOS data. GECOS is short for General Electric Comprehensive Operating Supervisor, a mainframe operating system. You may enter any text string in these fields, or nothing, though it is useful to include the user's full name. Study your */etc/passwd* file to see how other entries use the GECOS fields.

The *useradd* defaults are scattered across multiple configuration files; see Recipe 5.4 to learn how to change the defaults.

See Also

- *man 8 useradd*
- *man 5 login.defs*
- */etc/default/useradd*
- */etc/skel*
- */etc/login/defs*

5.3 Creating a System User with useradd

Problem

You want to create a system user with the *useradd* command.

Solution

The following example creates a new system user with no home directory, no login shell, and uses the correct UID numbering range for system users:

```
$ sudo useradd -rs /bin/false service1
```

-r means create a system user with a real ID in the correct numerical range for system users, and *-s* specifies the login shell. */bin/false* is a command that does nothing and prevents the user from logging into the system.

See the Discussion in Recipe 5.6 for information about UID and GID numbering.

Discussion

In olden times, most services ran as the *nobody* user. Now it is a common practice for services to have their own unique users, as this provides stronger security than the *nobody* user owning multiple services. You will rarely have to create a system user, as services should create their own unique users when they are installed.

The *nobody* user is always assigned UID 65534 and GID 65534.

See Also

- *man 8 useradd*
- *man 1 false*
- The Discussion in Recipe 5.6

5.4 Changing the useradd Default Settings

Problem

The default *useradd* settings are not right for you, and you want to change them.

Solution

The *useradd* configuration is spread across multiple configuration files: */etc/default/useradd*, */etc/login.defs*, and files in the */etc/skel* directory.

The following values appear in */etc/default/useradd*. This example shows the openSUSE defaults:

```
$ useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
```

GROUP=100 sets a single shared group as the default for all new users, traditionally *100*. The group must first exist, and *USERGROUPS_ENAB no* must be set in */etc/login.defs*. Then set *GROUP=* in */etc/default/useradd* to the GID of the user group. If our Duchess user is in a shared group, her *id* output shows *uid=1000(duchess) gid=100(users)*.

Enable private user groups by setting *USERGROUPS_ENAB yes* in */etc/login.defs*, then comment out *GROUP=* in */etc/default/useradd*. This creates a nonshared private group for each user. If our Duchess user has her own private group, her *id* output shows *uid=1000(duchess) gid=1000(duchess)*.

HOME=
: sets the default directory for all user home directories. The default is */home*.

INACTIVE=-1
: sets the number of days after a password expires until the account is locked. A value of 0 disables the account as soon as the password expires, and a value of –1 disables locking the account.

EXPIRE=
: sets an expiration date on the account, in YYYY-MM-DD format. For example, if you set it to 2021-12-31, the account will be disabled on that date. Leaving *EXPIRE=* empty means the account will not expire.

SHELL=/bin/bash
: sets the default command shell. */bin/bash* is the most commonly used Linux shell. Other values are any installed shell on the user's system, such as */bin/zsh* or */usr/bin/tcsh*. *cat /etc/shells* lists all installed shells.

SKEL=/etc/skel
: sets the location for the files that you want automatically distributed to new users. Most Linuxes put them in */etc/skel*. These are files such as *.bash_logout*, *.bash_profile* or *.profile*, *.bashrc*, and any other files you want new users to have. You may edit these files to suit your own requirements. *SKEL* is short for skeleton.

CREATE_MAIL_SPOOL=yes
: is a relic of olden times, and should be set to *yes*, as there may be some legacy processes that still need it.

The following values in */etc/login.defs* are relevant to user creation defaults:

- *USERGROUPS_ENAB yes* enables private user groups.
- *CREATE_HOME yes* configures *useradd* to automatically create private user home directories. This does not apply to system users (see Recipe 5.3).

Discussion

The UID numbering range is defined in */etc/login.defs*. Every UID must be unique, so the user account creation commands assign UIDs from the range defined in this file. Typically, human UIDs start at 1000, and are automatically assigned by *useradd*. You can override this with the *-u* option, but you must select an unused number that follows the configured numbering scheme (see the Discussion in Recipe 5.6).

A mandatory password change at first login is a simple precaution against the original password possibly falling into the wrong hands as it passes from the administrator to the user.

See Also

- *man 8 useradd*
- *man 5 login.defs*
- */etc/default/useradd*
- */etc/skel*
- */etc/login/defs*

5.5 Customizing the Documents, Music, Video, Pictures, and Downloads Directories

Problem

You followed Recipe 5.2 to create a new user, now you want to customize the Documents, Music, Video, Pictures, and Downloads directories for new users.

Solution

Creating these directories is not a function of *useradd*, but rather the X Desktop Group (XDG) user directories tool. The Documents, Music, Video, etc., directories are called the *well-known user directories*. These directories are set up from the */etc/xdg/user-dirs.defaults* configuration file, which establishes the default configuration for all users:

```
$ less /etc/xdg/user-dirs.defaults
# Default settings for user directories
#
# The values are relative pathnames from the home directory and
# will be translated on a per-path-element basis into the users locale
DESKTOP=Desktop
DOWNLOAD=Downloads
TEMPLATES=Templates
PUBLICSHARE=Public
DOCUMENTS=Documents
MUSIC=Music
PICTURES=Pictures
VIDEOS=Videos
# Another alternative is:
#MUSIC=Documents/Music
#PICTURES=Documents/Pictures
#VIDEOS=Documents/Videos
```

These are name-value pairs. The names cannot be changed. The values are the directories that the names are mapped to, and they are relative to users' home directories. For example, DOCUMENTS is mapped to */home/username/Documents*. The directories are created automatically for every new user when they start their graphical desktop environments for the first time. You may comment out any directories you wish to exclude or change the directories the names are mapped to.

Users may create their own personal configurations in *~/.config/user-dirs.dirs*. The directories must exist before applying the changes. The following example was created by our example user Duchess, who does not care for the boring default values. Note that the name-value pair syntax is different in *~/.config/user-dirs.dirs*:

```
XDG_DESKTOP_DIR="$HOME/table"
XDG_DOWNLOAD_DIR="$HOME/landing-zone"
XDG_DOCUMENTS_DIR="$HOME/omg-paperwork"
XDG_MUSIC_DIR="$HOME/singendance"
XDG_PICTURES_DIR="$HOME/piccies"
```

When your changes are complete and the new directories have been created, use the *xdg-user-dirs-update* command to apply your changes:

```
duchess@pc:~$ xdg-user-dirs-update --set DOWNLOAD $HOME/landing-zone
duchess@pc:~$ xdg-user-dirs-update --set DESKTOP  $HOME/table
duchess@pc:~$ xdg-user-dirs-update --set DOCUMENTS  $HOME/omg-paperwork
duchess@pc:~$ xdg-user-dirs-update --set MUSIC  $HOME/singendance
duchess@pc:~$ xdg-user-dirs-update --set PICTURES  $HOME/piccies
```

Log out, then log back in, and you will see something like Figure 5-1. XDG applies special icons to the well-known directories.

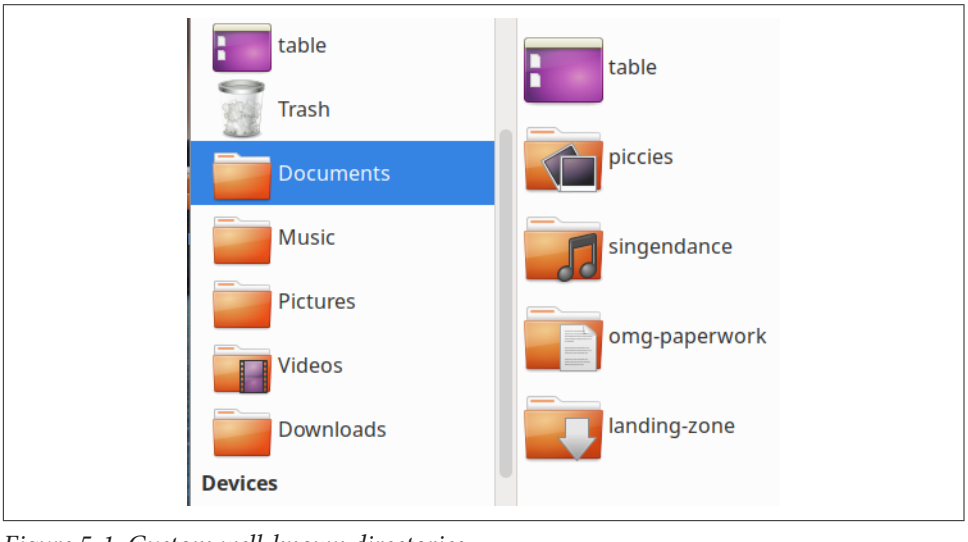


Figure 5-1. Custom well-known directories

The shortcuts in the side pane will not change, and the old directories are unchanged except they do not have the special icons. You will have to change the shortcuts and migrate the old directory contents manually.

Restore to the defaults in */etc/xdg/user-dirs.defaults* with this command:

```
$ xdg-user-dirs-update --force
```

Log out and back in to see the changes. Again, none of your directories are removed or changed in any way, except for the special icons that mark the well-known user directories.

Discussion

When you run the *xdg-user-dirs-update --set* command, you must use only the names as listed in *man 5 user-dirs.default*:

```
DESKTOP
DOWNLOAD
TEMPLATES
PUBLICSHARE
DOCUMENTS
MUSIC
PICTURES
VIDEOS
```

Only the values, which are the target directories, are configurable. Target directories must be relative to users' home directories. If you want to use directories outside of your home, create symlinks. For example, Duchess owns */users/stuff/duchess* and stores music files in it. The following example links this directory to */home/duchess/singendance*:

```
duchess@pc:~$ ln -s /users/stuff/duchess /home/duchess/singendance
```

See Also

- *man 5 user-dirs.defaults*
- *man 1 xdg-user-dirs-update*
- *man 5 user-dirs.conf*
- xdg-user-dirs at freedesktop

5.6 Creating User and System Groups with groupadd

Problem

You want to create groups with *groupadd*.

Solution

The following example creates a new user group *musicians*:

```
$ sudo groupadd musicians
```

Use *groupadd* with the *-r* option to create a system group:

```
$ sudo groupadd -r service1
```

Discussion

System groups differ from human user groups in the UID and GID numbering ranges assigned to them. This is configured in */etc/login.defs* for *groupadd* and *useradd*, as this example from Fedora 34 shows:

```
# Min/max values for automatic uid selection in useradd(8)
#
UID_MIN                  1000
UID_MAX                 60000
# System accounts
SYS_UID_MIN               201
SYS_UID_MAX               999
# Extra per user uids
SUB_UID_MIN                100000
SUB_UID_MAX             600100000
SUB_UID_COUNT               65536

#
# Min/max values for automatic gid selection in groupadd(8)
#
GID_MIN                  1000
GID_MAX                 60000
# System accounts
SYS_GID_MIN               201
SYS_GID_MAX               999
# Extra per user group ids
SUB_GID_MIN                100000
SUB_GID_MAX             600100000
SUB_GID_COUNT               65536
```

These define the number ranges available to the system administrator. All others are reserved for and managed by the system.

GID numbering is managed automatically by *groupadd*, according to the number ranges defined in */etc/login.defs* You may override this with the *-g* option, but your chosen GID must fall within the defined range, and must not already be used.

See Also

- *man 8 groupadd*
- */etc/login.defs*

5.7 Adding Users to Groups with usermod

Problem

You want to assign users to groups.

Solution

Use the *usermod* command. The following example adds Duchess to the *musicians* group:

```
$ sudo usermod -aG musicians duchess
```

This example adds Duchess to multiple groups:

```
$ sudo usermod -aG musicians,composers,stagehands duchess
```

Alternatively, you could edit */etc/group* and type Duchess's name after the appropriate group or groups. When you list multiple group members, the list must be comma-delimited, with no spaces between the names.

```
musicians:x:900:stash,madmax,duchess
```



Be Careful to Append, Not Replace

If you forget the *-a* option and use *-G* alone, all of the user's existing groups will be removed and replaced with the new groups. This is especially damaging if this removes users from their *sudo* group.

When you change group memberships for logged-in users, users must log out and then log back in to activate the changes. There are various workarounds to activate a new group assignment without logging out, but they all have limitations, such as being limited to the current shell. Groups are enumerated at login, so the most reliable solution is to log out and log back in.

Discussion

The *-a* option means append, and *-G* is group or groups.

See Also

- *man 8 usermod*

5.8 Creating Users with adduser on Ubuntu

Problem

You are running Debian or a Debian-based Linux, and need to know how to create new users with *adduser*.

Solution

adduser walks you through a complete new user setup, like this example for Stash Cat:

```
$ sudo adduser stash
Adding user 'stash' ...
Adding new group 'stash' (1009) ...
Adding new user 'stash' (1009) with group 'stash' ...
Creating home directory '/home/stash' ...
Copying files from '/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for stash
Enter the new value, or press ENTER for the default
        Full Name []: Stash Cat
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n]
```

Stash looks like this in */etc/passwd*:

```
stash:x:1009:1009:Stash Cat,,,:/home/stash:/bin/bash
```

Discussion

The *adduser* defaults are managed in */etc/adduser.conf*. This provides a number of useful defaults such as:

DSHELL=
: sets the default login shell. */bin/bash* is the most commonly used Linux shell. Other values are any installed shell on the user's system, such as */bin/zsh* or */usr/bin/tcsh*. *cat /etc/shells* lists all installed shells.

USERGROUPS=yes
: creates user private groups, *no* puts all users into the same group.

USERS_GID=100
: is required when *USERGROUPS=no* is set.

EXTRA_GROUPS=
: is your list of supplemental groups for new users, for example, *EXTRA_GROUPS="audio video plugdev libvirt"*.

ADD_EXTRA_GROUPS=1
: makes the groups listed in *EXTRA_GROUPS=* the default for new users.

/etc/adduser.conf contains the following user and group numbering scheme:

```
FIRST_SYSTEM_UID=100
LAST_SYSTEM_UID=999

FIRST_SYSTEM_GID=100
LAST_SYSTEM_GID=999

FIRST_UID=1000
LAST_UID=59999

FIRST_GID=1000
LAST_GID=59999--
```

Fedora Linux includes *adduser*, but it is not really *adduser*, just a symlink to *useradd*:

```
$ stat /usr/sbin/adduser
  File: /usr/sbin/adduser -> useradd
  Size: 7    Blocks: 0    IO Block: 4096   symbolic link
[...]
```

See Also

- *man 5 adduser.conf*

5.9 Creating a System User with adduser on Ubuntu

Problem

You want to create a system user with *adduser* on your Ubuntu (or Debian, Mint, or other Debian derivative) system.

Solution

The following example creates a new system user, *service1*, with *adduser*, without a home directory, and with its own unique primary group:

```
$ sudo adduser --system  --no-create-home --group service
Adding system user 'service1' (UID 124) ...
Adding new group 'service1' (GID 135) ...
Adding new user 'service1' (UID 124) with group 'service1' ...
Not creating home directory '/home/service1'.
```

This is how it looks in */etc/passwd*:

```
service1:x:124:135::/home/service1:/usr/sbin/nologin
```

Discussion

System users do not have home directories.

Back in olden times, it was common for services to run as the *nobody* user and group, except Debian which uses *nobody* and *nogroup*. Recycling the same user for multiple services is a security weakness. It is unlikely you will ever have to create a system user, as the common practice now is for the package installer to create a unique user and group when you install a new service. But now you know how, just in case you ever need to.

nobody and *nogroup* always have a real ID of 65534.

See Also

- *man 8 adduser*

5.10 Creating User and System Groups with addgroup

Problem

You want to know how to create user and system groups with Debian's *addgroup* command.

Solution

The following example creates a human user group:

```
$ sudo addgroup composers
Adding group 'composers' (GID 1010) ...
Done.
```

It looks like this in */etc/group*:

```
composers:x:1010:
```

This example creates a new system group:

```
$ sudo addgroup --system service1
Adding group 'service1' (GID 136) ...
Done.
```

Discussion

The difference between user and system groups is they each have different numbering ranges for UIDs and GIDs, which are configured in */etc/adduser.conf*.

See Also

- *man 8 addgroup*

5.11 Checking Password File Integrity

Problem

There is a lot going on in all these user files and group files, and you want to know if there is some kind of checker to verify that these files are written correctly.

Solution

The *pwck* command checks the integrity of */etc/passwd* and */etc/shadow*, and *grpck* checks */etc/group* and */etc/gshadow*. They look for correct format, valid data, valid names, and valid GIDs (see the man pages for a complete list). When you run these with no options, they report both warnings and errors:

```
$ sudo pwck
user 'news': directory '/var/spool/news' does not exist
user 'uucp': directory '/var/spool/uucp' does not exist
user 'www-data': directory '/var/www' does not exist
user 'list': directory '/var/list' does not exist

$ sudo grpck
group mail has an entry in /etc/gshadow, but its password field in /etc/group is
not set to 'x'
grpck: no changes
```

Add the **-q** option to report only errors:

```
$ sudo pwck -q

$ sudo grpck -q
group mail has an entry in /etc/gshadow, but its password field in /etc/group
is not set to 'x'
```

This shows an error in */etc/gshadow*. This is not a very helpful message because it's not really an error. It is not a common practice to place passwords on user groups, so reporting this an as error is needlessly confusing. The other checks are useful, such as the correct number of fields, and a unique valid group name.

You will never edit */etc/shadow* or */etc/gshadow*, but only */etc/passwd* and */etc/group*.

Discussion

The following example shows an error that must be corrected. Type **n**, when prompted, to prevent deleting the entries. The first "delete line" example is from */etc/passwd*, and the second is in */etc/shadow*:

```
$ sudo pwck -q
invalid password file entry
delete line 'fakeservice:x:996:996::/home/fakeservice'? n
delete line 'fakeservice:!:18469::::::'? n
pwck: no changes
```

Then correct the line in */etc/passwd*, and that will fix both error messages. In the example, *fakeservice:x:996:996::/home/fakeservice* is missing the last field, and should be *fakeservice:x:996:996::/home/fakeservice:/bin/false*.

The "directory does not exist" warnings for */etc/passwd* usually refer to default system users that are not in use. For example:

```
user 'www-data': directory '/var/www' does not exist
```

The *www-data* user is unused when you are not running an HTTP server, and there is no */var/www* directory until you install an HTTP server.

The "no changes" message means that no changes were made to the password file.

See the man pages for a complete list of checks.

See Also

- *man 8 pwck*
- *man 8 grpck*

5.12 Disabling a User Account

Problem

You want to disable a user account without deleting it.

Solution

To temporarily deactivate an account, disable the user's password with the *passwd* command:

```
$ sudo passwd -l stash
passwd: password expiry information changed.
```

Now the user cannot log in. The following example unlocks the user's account:

```
$ sudo passwd -u stash
passwd: password expiry information changed.
```

This does not prevent a user from logging in via a different authentication method, such as an SSH key. To completely disable a user account, use *usermod*:

```
$ sudo usermod --expiredate 1 stash
```

When the user tries to log in, they see a "Your account has expired; please contact your system administrator" message. Restore their account:

```
$ sudo usermod --expiredate -1 stash
```

Discussion

Another way to disable a user is to replace the x in the password field in */etc/passwd* with an asterisk (*):

```
stash:*:1009:1009:Stash Cat,,,:/home/stash:/bin/bash
```

Reenable Stash by replacing the asterisk with *x*.

See Also

- *man 1 passwd*

5.13 Deleting a User with userdel

Problem

You need to delete a user, and possibly their home directory and its contents.

Solution

The following example uses the *userdel* command to delete the user Stash from */etc/passwd*, Stash's primary group and all group memberships, and the shadow files:

```
$ sudo userdel stash
```

If Stash belongs to a shared primary user group (discussed in Recipe 5.4), the group will not be deleted.

Use the *-r* option to delete the user's home directory and its contents, and their mail spool:

```
$ sudo userdel -r stash
```

If the user owns files outside of their home directory, you will have to find and take care of them separately (see Recipe 5.16).

Discussion

Read your */etc/passwd* and */etc/group* files before and after deleting a user to see the user disappear.

It is a good practice to clean up after removing a user.

See Also

- *man userdel*

5.14 Deleting a User with deluser on Ubuntu

Problem

You're running Ubuntu (or another Debian derivative) and want to use *deluser* to delete a user.

Solution

The following example deletes the user Stash from */etc/passwd*, Stash's primary group from */etc/group*, and their corresponding shadow files:

```
$ sudo deluser stash
Removing user 'stash' ...
Warning: group 'stash' has no more members.
Done.
```

deluser will not remove the primary group of an existing user, so if Stash belongs to a shared primary group it will not be removed.

This example removes Stash's home directory and makes a backup of all the deleted files:

```
$ sudo deluser --remove-all-files --backup stash
```

Discussion

--backup creates a compressed archive of the user's files in the current directory. Use the *--backup-to* option to select a different directory:

```
$ sudo deluser --remove-all-files --backup-to /user-backups stash
```

If the user owns files outside of their home directory, you will have to hunt them down and deal with them manually (see Recipe 5.16).

See Also

- *man 8 deluser*

5.15 Removing a Group with delgroup on Ubuntu

Problem

You have an Ubuntu system and want to use the *delgroup* command to delete groups.

Solution

The following example removes the *musicians* group:

```
$ sudo delgroup musicians
```

delgroup will not remove the primary group of an existing user. It will remove supplemental groups even when they have members. If you do not want to remove groups that have members, use the *--only-if-empty* option:

```
$ sudo delgroup --only-if-empty musicians
```

Discussion

The default behavior of *delgroup* is configured in */etc/deluser.conf* and */etc/adduser.conf*.

See Also

- *man 8 delgroup*

5.16 Finding and Managing All Files for a User

Problem

You want to delete a user, but you don't want a bunch of orphaned files left over, and you need to find all of them.

Solution

The *find* command will locate all files on the local system by UID or GID. The following example searches the entire root directory for all files owned by the user's UID:

```
$ sudo find / -uid 1007
```

This can take some time if there are lot of files to search. If you are certain you do not have to search the entire filesystem, you can narrow your search to specific subdirectories, such as */etc*, */home*, or */var*:

```
$ sudo find /etc -uid 1007
$ sudo find /home -uid 1007
$ sudo find /var -uid 1007
```

You may also search by GID, username, or group name:

```
$ sudo find / -gid 1007
$ sudo find / -name duchess
$ sudo find / -group duchess
```

Now that you know where all the files are, what to do with them? One option is to change their ownership to another user and let the new user deal with them:

```
$ sudo find /backups -uid 1007 -exec chown -v 1010 {} \;
changed ownership of '/backups/duchess/' from 1007 to 1010
changed ownership of '/backups/duchess/bin' from 1007 to 1010
changed ownership of '/backups/duchess/logs' from 1007 to 1010
```

You could combine *find* and *cp* to find and copy all the files to a different directory:

```
$ sudo find / -uid 1007 -exec cp -v {} /orphans \;
```

Using *cp -v* prints progress messages and copies only the files and not their parent directories. If you wish to copy the parent directories, use the *-r* option:

```
$ sudo find / -uid 1007 -exec cp -rv {} /orphans \;
```

Copying leaves the original files in place. After they are safely copied, you may wish to delete the originals. One way to do this is to run *find* again and use *rm* to delete the original files:

```
$ sudo find / -uid 1007 -exec rm -v {} \;
```

This deletes the files but not the directories. Use the *-r* option to delete the directories if you are certain there are no other files in those directories that you want to keep:

```
$ sudo find / -uid 1007 -exec rm -rv {} \;
```

One more option is to use *find* and *mv* to move the files to a different location:

```
$ sudo find / -uid 1007 -exec mv {} /orphans \;
```

If you see a "No such file or directory" message, usually that is because the file or directory was moved, which you can verify by looking in the directory they were moved to.

Find files owned by a nonexistent user or group:

```
$ find / -nouser
$ find / -nogroup
```

Discussion

Be careful with *mv* and *rm* because there is no undo. If you make a mistake, your best hope of recovery is from backup.

Cleaning up after departed users can be a chore, because computers make it too easy to create as many files as your storage will hold. If you find yourself thinking that *find* is taking too long, keep in mind it will find everything while you go do something else.

See Also

- *man 1 find*
- *man 1 mv*
- *man 1 cp*
- *man 1 rm*

5.17 Using su to Be Root

Problem

You need to know how to get root permissions to perform some administration chores.

Solution

Use the *su* command to change to the root user when you need to do system chores:

```
duchess@pc:~$ su -l
Password:
root@pc:~#
```

If you do not know root's password, or if there is no root password, see Recipe 5.21 to learn how to use *sudo* to set a root password.

When you are finished, exit root and return to your own account:

```
root@pc:~# exit
logout
duchess@pc:~$
```

The *-l* option invokes the root user's environment, changing to root's home directory and loading root's environment variables. Omit *-l* to keep your own environment:

```
duchess@pc:~$ su
Password:
root@pc:/home/duchess~#
```

Discussion

You can change to any user, as long as you have their password.

Using *su* to change to root gives you absolute power over your system, and every command that you run is run as root. Consider using *sudo* (see Recipe 5.18), which provides some safety features, such as protecting root's password and leaving an audit trail.

See Also

- *man 1 su*

5.18 Granting Limited Root Powers with sudo

Problem

You want to delegate some system administration chores to other users, and you want to limit their root powers to what is needed for their specific tasks.

Solution

Use the *sudo* command. *sudo* is safer than *su* because it grants limited root powers to specific users for specific tasks, logs activity, and caches the user's password for a limited amount of time, with a default of 15 minutes. After 15 minutes, the user must provide *sudo* with their password again. The caching duration is configurable. *sudo* protects root's password because *sudo* users use their own passwords.



Some Linux distributions, such as openSUSE, default to *sudo* asking for the root user's password. See Recipe 5.22 to learn how to change this.

/etc/sudoers is the configuration file, and you should edit it with a special command, *visudo*. This opens */etc/sudoers* with your default text editor, and you can review and edit the default configuration. Once again, Duchess demonstrates for us:

```
duchess@pc:~$ sudo visudo
[sudo] password for duchess:
[...]
##Allow root to run any commands
root    ALL=(ALL) ALL

# Allow members of group sudo to execute any command
```

```
%sudo   ALL=(ALL) ALL
[...]
```

%sudo ALL=(ALL) ALL means that any user you add to the *sudo* group gets full *sudo* powers just like root. The percent sign indicates *%sudo* is a group from */etc/group*, and not a group configured in */etc/sudoers*.

Suppose you have a junior admin, Stash, whose job is installing and removing software and keeping the system updated. You could create a system group for Stash. Or you could configure Stash for this task in */etc/sudoers*. The following example gives Stash *sudo* powers to run the listed commands. You need the username, the hostname of the local machine, and a comma-delimited list of allowed commands:

```
stash server1 = /bin/rpm, /usr/bin/yum, /usr/bin/dnf
```

Suppose you want to give Stash more admin chores, such as managing services. The allowed commands list will get long, so you could create some command aliases instead. The following example aliases the software management commands to SOFTWARE, and the service management commands to SYSTEMD:

```
Cmnd_Alias SOFTWARE = /bin/rpm, /usr/bin/yum, /usr/bin/dnf
Cmnd_Alias SYSTEMD = /usr/bin/systemctl start, /usr/bin/systemctl stop,
/usr/bin/systemctl reload, /usr/bin/systemctl restart, /usr/bin/systemctl
status, /usr/bin/systemctl enable, /usr/bin/systemctl disable,
/usr/bin/systemctl mask, /usr/bin/systemctl unmask
```

Now Stash's configuration looks like this:

```
stash server1 = SOFTWARE, SYSTEMD
```

You may create user groups in */etc/sudoers* (not related to system groups in */etc/group*), then assign them some command aliases:

```
User_Alias  JRADMIN = stash, madmax

JRADMIN server1 = SOFTWARE, SYSTEMD
```

You may create a *Host_Alias* to give a user *sudo* rights on multiple machines:

```
Host_Alias SERVERS = server1, server2, server3
```

Then bring in the JRADMINs:

```
JRADMIN SERVERS = SOFTWARE, SYSTEMD
```

Discussion

When your limited *sudo* users try to run a not-allowed command, they see this message: "Sorry, user *duchess* is not allowed to execute `/some/command` as root on *server2*."

Don't put too much faith in limiting users to a specific set of commands. Many everyday applications provide a means for privilege escalation via shell escape, and your users can gain full root powers. This example shows how it works with *awk*:

```
$ sudo awk 'BEGIN {system("/bin/bash")}'
root@client4:/home/duchess#
```

And just like that, Duchess has full root powers. The humble *less* command also provides a shell escape. Read a file with *less* that is large enough to require paging:

```
$ sudo less /etc/systctl.conf
#
# /etc/sysctl.conf - Configuration file for setting system variables
# See /etc/sysctl.d/ for additional system variables.
# See sysctl.conf (5) for information.
/etc/sysctl.conf
```

Type **`!sh`**, then when the prompt changes, type **`whoami`**:

```
duchess@client4:~$ sudo less /etc/systctl.conf
#
# /etc/sysctl.conf - Configuration file for setting system variables
# See /etc/sysctl.d/ for additional system variables.
# See sysctl.conf (5) for information.
!'sh'
duchess@client4:~$ sudo less /etc/sysctl.conf
# whoami
root
```

Type **`exit`** to return your normal shell.

In my experience, it is extremely difficult to keep track of the many applications that can provide a shell escape. *journalctl* records everything, should you wish to monitor your *sudo* users (see Recipe 20.1).

In some Linuxes, such as Fedora, the *wheel* group is the default *sudo* group. Check your */etc/sudoers* file to see how your distribution configures this. You can also create your own *sudo* group, and name it whatever you want.

The */etc/sudoers* file controls users only on the local machine. Including other machines, like the SERVERS alias, allows you to share a single configuration file on multiple machines. *sudo* ignores any items, such as hosts or users, that are not present on the local machine.

Let's dissect *root ALL=(ALL) ALL* to understand what all those ALLs mean.

root
: is in the user field, and this field holds any single user, user alias, or system group.

ALL=
: is in the host field. ALL means any host anywhere, or you could use a host alias, or name a single host.

(ALL)
: is in the optional users field. *(ALL)* means the user or users can run commands as any other user, or you can specify certain users.

ALL
: in is the command field. *ALL* is unrestricted, or you can specify a list of allowed commands.

See Also

- *man 8 sudo*
- *man 5 sudoers*

5.19 Extending the sudo Password Timeout

Problem

On most Linux distributions, *sudo* caches passwords for a default interval of 15 minutes. Then after 15 minutes you have to enter your password again. You are tired of having to enter your password so often when you have a lot of work to do, and want to make the caching interval longer.

Solution

Change the caching interval in */etc/sudoers*. Open the file for editing with *visudo*:

```
$ sudo visudo
```

Then look for a *Defaults* line and set your new cache interval. The following example sets it to 60 minutes:

```
$ Defaults timestamp_timeout=60
```

If you set it to 0, *sudo* asks for your password every time you use it.

If you set *timestamp_timeout* to a negative number, like *-1*, your password never expires.

Discussion

The *sudo* password caching is a useful protection against accidents, such as forgetting you are running as root or wandering away and allowing someone else to have some fun with your computer.

See Also

- *man 8 sudo*
- *man 5 sudoers*

5.20 Creating Individual sudoers Configurations

Problem

You want to set some different *sudo* configurations for your users; for example, you want your junior admins to have a different password timeout than you. Yours is long, and you want theirs to be short.

Solution

You may create individual configurations in */etc/sudoers.d*. The following example creates a 30-minute password timeout for Stash:

```
$ cd /etc/sudoers.d/
$ sudo visudo -f stash
```

Type **`Defaults timestamp_timeout=30`**, save the file, and you're done. You can see the new file:

```
$ sudo ls /etc/sudoers.d/
README stash
```

You only need to enter configuration items that are different from the entries in */etc/sudoers*, not to replicate the whole file.

Discussion

This is a nice feature for managing multiple users. Instead of managing one big configuration file, break it up into smaller per-user files.

See Also

- *man 8 sudo*
- *man 5 sudoers*

5.21 Managing the Root User's Password

Problem

Your Linux distribution set you up as system administrator during installation, with unrestricted *sudo* privileges, and did not create a root password. Or, your root user had a password but you forgot it. You need to know how to give root a new password.

Solution

When you want to run as "real" root, use *sudo* to *su* to root:

```
duchess@pc:~$ sudo su -l
[sudo] password for duchess:
root@pc:~#
```

At this point you can use the *passwd* command to give root a password so you can log in as root directly, or reset a lost root password.

Discussion

There are times when you need a root password and not *sudo*; for example, when you boot to an emergency runlevel.

See Also

- *man 8 sudo*
- *man 5 sudoers*
- *man 1 passwd*

5.22 Changing sudo to Not Ask for the Root Password

Problem

You want your *sudo* users to authenticate with their own passwords, but your Linux system asks for the root user's password, like the following example:

```
$ sudo visudo
[sudo] password for root:
```

Solution

This is the default behavior on some Linux distributions, such as openSUSE.

When you install Ubuntu Linux and make your user an administrator during installation, Ubuntu configures your user appropriately with full *sudo* powers, equivalent to root but using your own password. openSUSE does not, but instead configures your user to use the password of the target user, which is root.

To set up *sudo* users to always be asked for their own passwords, edit */etc/sudoers* by commenting out these two lines:

```
duchess@pc:~$ sudo visudo
```

```
# Defaults targetpw
# ALL   ALL=(ALL) ALL
```

In openSUSE and Fedora, create *sudo* users with full root powers by adding them to the *wheel* group in */etc/group*. (For limited users, refer to Recipe 5.18.)

The change takes effect immediately after saving your changes and closing the file.

Discussion

Protecting the root user's password is a primary reason to use *sudo*, rather than *su*.

See Also

- Recipe 5.18

CHAPTER 6

Managing Files and Directories

Linux provides strong basic controls for access to files and directories with configurable privileges. Every file and directory has three levels of ownership, including user, group, and other; and multiple levels of access, including read, write, and execute. You can protect your personal files and control who has access to them, and the root user can manage access to commands, scripts, shared files, and system files.

Even when you are using stronger access control tools—tools such as SELinux or AppArmor—it is still important to get the fundamentals right.

On a Linux system, both human users and system services have user accounts. Some system services need user accounts to control privileges, just like human users.

Every file has three types of ownership: owner, group, and other (sometimes *other* is expressed as *world*). The owner is a single user, the group owner is a single group, and other is everyone else who has access to the file.

Every file has six permission modes—read, write, and executable—and three special modes: the *sticky bit*, *setuid*, and *setgid*.

File permissions control which users can create, read, edit, or delete a file, and which users can execute a command. The special modes control who can move, delete, or rename a file, and who can execute a command with elevated privileges.

Directory permissions control which users can edit or enter a directory and who can read, edit, add, or remove files from a directory.

Remember the fundamental Linux security principle: use the minimum necessary privileges to get the job done.



Limitations of Privileges

Anyone who can read a file can copy it.

You cannot prevent the root user, or *sudo* users with sufficient privileges, from accessing your files.

Permissions and ownership are functions of filesystems and can be bypassed by reading a storage device from another Linux instance, such as booting up a live Linux from removable media to access the host system, or removing the hard drive and connecting it to a different machine. You only need root privileges on the system that you mount the storage device on, and do not need to know anything about the original file owners and permissions.

On a Linux system the root user, also called the superuser, reigns supreme. Root can do almost anything, including editing and deleting other users' files, entering any directory, and running any command. Normal, or unprivileged, users may temporarily assume root powers with the *sudo* or *su* commands (see Recipes 5.17 and 5.18).

Every user has a unique identification (UID), and belongs to at least one group (see Recipe 5.1). Every user in a group shares the permissions of that group.

To see what all of this looks like, take a look at */etc*, which contains system configuration files:

```
$ stat --format=%a:%A:%U:%G /etc
755:drwxr-xr-x:root:root
```

The command output shows the directory's *mode*, or set of permissions, in two forms, *755:drwxr-xr-x*. *755* is octal notation, and *drwxr-xr-x* is symbolic notation. These are two different ways of expressing the same mode, which in this example is unrestricted privileges for the directory owner, and group and other may only enter the directory. File modes are discussed in detail in this chapter.

root:root is the owner and group. Files and directories can have different owners and groups; for example, */etc/cups* is owned by *root:lp*.

In this chapter you will learn about the special modes: the *sticky bit*, *setuid*, and *setgid*. The setuid and setgid modes elevate user and group permissions to the same level as the file owner. These are used only in special cases, and used very carefully because privilege escalation is a potential security risk. The sticky bit prevents anyone but the file owner, or anyone with root privileges, from deleting, renaming, or moving files they do not own in a directory, such as */tmp*.

You will learn how to set ownership and modes, create and delete files and directories, configure default privileges, transfer file ownership to a different user or group, and copy, move, and rename files and directories.



Using sudo

Most of the examples in this recipe use the dollar sign command prompt, $, which indicates an unprivileged user. Depending on your own file permissions, you may need *sudo* for some operations.

6.1 Creating Files and Directories

Problem

You want to organize your files by placing them in directories.

Solution

Use the *mkdir* command to create directories. The following example creates a new subdirectory in the current directory:

```
$ mkdir -v presentations
mkdir: created directory 'presentations'
```

Create a subdirectory two levels down inside the current directory, and its parent directories, with the *-p* (parent) option:

```
$ mkdir -p presentations/2020/august
mkdir: created directory 'presentations/2020'
mkdir: created directory 'presentations/2020/august'
```

Create a new top-level directory, which is relative to root, /. You need root privileges to do this:

```
$ sudo mkdir -v /charts
mkdir: created directory '/charts'
```

You can set permissions when you create a directory:

```
$ mkdir -m 0700 /home/duchess/dog-memes
```

Files are created by applications, such as word processors and image editors, and special commands like *touch*. The *touch* command creates a new empty file:

```
$ touch newfile.txt
```

See Recipe 6.2 to learn how to use *touch* to quickly create batches of files for testing.

Discussion

If you are having trouble visualizing file trees, and how all directories are relative to /, try the *tree* command. Root, /, is at the top:

```
$ tree -L 1 /
/
├── backups
```

```
├── bin
├── boot
[...]
```

You have probably noticed that this is upside down. In the real world, trees branch from the root, but the *tree* command displays the directory tree branching downward. There is a reason for this: we read screens from the top down.

This example lists only the top-level directories under root. *-L 2* shows second-level directories, *-L 3* goes to three levels, and so on.

See Also

- Recipe 6.2
- *man 1 mkdir*
- *man 1 touch*
- *man 1 yes*
- *man 1 tree*

6.2 Quickly Creating a Batch of Files for Testing

Problem

You want to create batches of files to use for testing file permissions, and for any testing that needs a lot of files in a hurry.

Solution

Use the *touch* command. The following example creates a single new empty file:

```
$ touch newfile.txt
```

Create 100 new empty files:

```
$ touch file{00..99}
```

This creates 100 new files named *file00*, *file01*, *file02*, and so on. You may give them file extensions and name them anything:

```
$ touch test{00..99}.doc
$ ls
test00.doc
test01.doc
test02.doc
[...]
```

Put the numbers first in the filename for easy ordering:

```
$ touch {00..99}test.doc
$ ls
00test.doc
01test.doc
02test.doc
[...]
```

A fast way to populate the files with content is to use the *yes* command. The following example creates a 500 MB file filled with the repeated line "This is a test file":

```
$ yes This is a test file | head -c 500 MB > testfile.txt
```

Create a batch of 100 files with 1 MB of content in each file:

```
$ for x in {01..100};
> do yes This is a test file | head -c 1MB > $x-testfile.txt;
> done
```

The new files look like this:

```
001-testfile.txt
002-testfile.txt
003-testfile.txt
[...]
```

Discussion

You may customize this command in a number of ways: filenames, file sizes, numbering, and the text for *yes*.

The examples in the recipe pad the numbers in the filenames with leading zeroes so they will order correctly. Most graphical file managers handle ordering numbered filenames correctly, but the default for *ls* is lexicographic order. The following example demonstrates this with a 1- to 3-digit numbering range:

```
$ touch {0..150}test.doc
$ ls -C1
0test.doc
100test.doc
101test.doc
102test.doc
103test.doc
104test.doc
105test.doc
106test.doc
107test.doc
108test.doc
109test.doc
10test.doc
110test.doc
111test.doc
```

```
112test.doc
113test.doc
114test.doc
115test.doc
116test.doc
117test.doc
118test.doc
119test.doc
11test.doc
120test.doc
121test.doc
[...]
```

Lexicographic ordering treats the filenames as text strings instead of integers and characters, and compares each number and letter individually, from left to right. Lexicographic ordering doesn't know that 10 is smaller than 100, only that 101 follows 100, 102 follows 101, and 10t follows 109 because letters follow numbers, so the *t* follows the *9*.

You can use leading zeroes to make all the numbers the same number of characters, or list your files with *ls -v*. This treats the numbers in filenames as integers and not characters, so they are listed in correct numerical order.

See Also

- *man 1 ls*
- *man 1 touch*
- *man 1 yes*

6.3 Working with Relative and Absolute Filepaths

Problem

You need to understand the difference between relative and absolute filepaths, and how to find where you are in the filesystem.

Solution

Absolute filepaths always start at the root, /, such as */boot* and */etc*. Relative filepaths are relative to your current directory and do not have a leading slash. Suppose you are in your home directory and it contains the following subdirectories:

```
madmax@client2:~$ ls --group-directories-first
 Audiobooks
 bin
 Desktop
```

```
Documents
Downloads
games
Music
Pictures
Public
Templates
Videos
```

In this example, the absolute path to *Audiobooks* is */home/madmax/Audiobooks*, and the relative path is *Audiobooks*. Use the *cd* command to enter this directory with either the absolute path:

```
$ cd /home/madmax/Audiobooks
```

Or the relative path:

```
$ cd Audiobooks
```

The directory you are in is the current working directory, *cwd*. Confirm your *cwd* with the *pwd* (print working directory) command:

```
$ pwd
/home/madmax
```

Discussion

Absolute and relative filepaths are a common source of confusion. Remember that when the filepath begins with a slash (/), it is an absolute path. When there is no leading slash, it is relative to your current working directory.

Some applications and commands require relative paths; for example, *rsync include* and *exclude* lists use filepaths that are relative to the directories being copied.

See Also

- *man 1 pwd*
- Chapter 7

6.4 Deleting Files and Directories

Problem

You had fun creating a bunch of files and directories, and now you want to get rid of them.

Solution

Use the *rm* (remove) command with caution, because *rm* will happily delete everything you tell it to, so be sure you tell it the right files or directories to delete.

Delete a single file, with verbose output:

```
$ rm -v aria.ogg
removed 'aria.ogg'
```

Use the *-i* flag to prompt for confirmation first:

```
 $ rm -iv intermezzo.wav
rm: remove regular file 'intermezzo.wav'? y
removed 'intermezzo.wav'
```

Add the *-r* (recursive) flag to delete a directory and all of its files and subdirectories. Combining *-r* with *-i* will prompt you for confirmation before each deletion:

```
$ rm -rvi rehearsals
rm: descend into directory 'rehearsals'? y
rm: remove regular file 'rehearsals/brass-section'? y
[...]
```

If you are confident you don't need to be prompted for every deletion, omit the *-i* option.

This example deletes only the *jan* subdirectory:

```
$ rm -rv rehearsals/2020/jan
```

This example deletes the *rehearsals* directory and all of its files and subdirectories:

```
$ rm -rv rehearsals
```

Use wildcards to match file names to delete, for example by file extension:

```
$ rm -v *.txt
```

Or by files named with the same text strings:

```
$ rm -v aria*
```

If *rm* refuses to delete a file or directory, and you are certain you want to delete it, add the *-f* (force) option.

Discussion

rm -rf / will erase your entire root filesystem (if you have root privileges). Some folks think it is a funny prank to tell newbies to do this. It is not funny. It is fun to run it on a test machine, or on a virtual machine, and observe how long the system keeps running because processes in memory are still running, even though the filesystem is erased from disk.

See Also

- *man 1 rm*

6.5 Copying, Moving, and Renaming Files and Directories

Problem

You have directories, and you have files. You want to move files into the directories, change filenames, and make copies.

Solution

Use the *cp* command for copying, and the *mv* command for moving or renaming.

This example copies two files from the current working directory into the *~/songs2* directory:

```
$ cp -v aria.ogg solo.flac ~/songs2/
'aria.ogg' -> '/home/duchess/songs2/aria.ogg'
'solo.flac' -> '/home/duchess/songs2/solo.flac'
```



The Tilde Represents Your Home Directory

The tilde is short for your home directory, so in the example, *~/songs2* is the same as */home/duchess/songs2/*.

Copy a directory and all of its contents with the *-r* (recursive) option:

```
$ cp -rv ~/music/songs2 /shared/archives
```

The recursive example only copies the directory and its files. Use the *--parents* option to preserve parent directories. The following example copies *songs1* and its contents, and preserves the filepath *duchess/music/songs2/*:

```
$ cp -rv --parents duchess/music/songs2/ shows/
duchess -> shows/duchess
duchess/music -> shows/duchess/music
'duchess/music/songs2' -> 'shows/duchess/music/songs2'
'duchess/music/songs2/intro.flac' -> 'shows/duchess/music/songs2/intro.flac'
'duchess/music/songs2/reprise.flac' -> 'shows/duchess/music/songs2/reprise.flac'
'duchess/music/songs2/solo.flac' -> 'shows/duchess/music/songs2/solo.flac'
```

The other contents of *duchess* and *music* are not copied, only *songs2* and its contents.

Use the *mv* command to move and rename files. This example moves two files to another directory:

```
$ mv -v aria.ogg solo.flac ~/songs2/
renamed 'aria.ogg' -> '/home/duchess/songs2/aria.ogg'
renamed 'solo.flac' -> '/home/duchess/songs2/solo.flac'
```

The following example moves a directory into another directory:

```
$ mv -v ~/songs2/ ~/music/
```

Discussion

Some useful *cp* options are:

- *-a, --archive* preserves all the file attributes, such as mode, ownership, and timestamps.
- *-i, --interactive* prompts before overwriting destination files.
- *-u, --update* overwrites an existing destination file only if the source file is newer. This saves time when you're recopying a batch of files, and some of the copies are unchanged. (*rsync* is better for efficient file transfers by copying only changes, see Chapter 7.)

mv has some useful options:

- *-i, --interactive* prompts before overwriting destination files.
- *-n, --no-clobber* prevents overwriting destination files.
- *-u, --update* moves your files only when they are newer than the destination files, or when they are moved for the first time.

See Also

- *man 1 cp*
- *man 1 mv*

6.6 Setting File Permissions with chmod's Octal Notation

Problem

You know that the *chmod* (change mode) command supports both octal and symbolic notation, and you want to use octal notation to manage file permissions.

Solution

The following examples show how to set different permissions on files using octal notation. The first example grants read-write access to the owner of the *file.txt* file, and excludes all access for group and world:

```
$ chmod -v 0600 file.txt
mode of 'file.txt' changed from 0644 (rw-r--r--) to 0600
(rw-------)
```

The file owner can read, edit, and delete the file, while other users can do nothing with it, not even read it, though they can see it listed in a file manager.

Make a file world readable and writeable, allowing everyone to do whatever they want to it:

```
$ chmod 0666 file.txt
```

In the next example, *file.txt* is changed to read-write for the file owner and read-only for group and world:

```
$ chmod -v 0644 file.txt
mode of 'file.txt' changed from 0666 (rw-rw-rw-) to 0644 (rw-r--r--)
```

A common permission set is to give the owner and group the same permissions, such as read-write, and to exclude other:

```
$ chmod 0660 file.txt
```

Commands and scripts require the executable bit to be set. This example makes the *backup.sh* script executable and read-write for the owner, executable and readable for group, and inaccessible to other:

```
$ chmod 0750 backup.sh
```

Octal notation has four fields, but you probably will use the last three fields the most often, and the first field rarely. The first field is reserved for the special modes (see Recipe 6.8).

Discussion

Octal notation uses integers 0-7. Table 6-1 shows the relationship between owners and permissions.

Table 6-1. Octal fields

Mode	Owner	Group	Other
Read	4	4	4
Write	2	2	2
Execute	1	1	1
No permission	0	0	0

A file or directory has one user owner, and one group owner. *Other* is everyone else. A directory or executable that is unrestricted to everyone is mode 0777, and an unrestricted file is mode 0666.

When you're not familiar with Linux file permissions, it might help to see them in another view, like in Table 6-2.

Table 6-2. Linux file permissions

Permission	Description
7	Read, write, execute. Directories differ from files because all directories require the executable bit set. You can assign a directory any permissions, just like a file, but without the executable bit no one can enter the directory (with the *cd* command or in a file manager). Scripts and binary commands must have the executable bit set, or they will be treated as ordinary files.
6	Read and write.
5	Read and execute. This is a common permission for commands.
4	Read.
3	Write and execute.
2	Write.
1	Execute.
0	No permission.

See Also

- *man 1 chmod*
- Recipe 6.8

6.7 Setting Directory Permissions with chmod's Octal Notation

Problem

You know that permissions are managed a little differently on directories, and you want to manage them with chmod's octal notation.

Solution

Directories must have the executable bit set. This might sound a little strange, but it is necessary for entering the directory with the *cd* command or with a file manager.

The following examples creates a shared directory:

```
$ sudo mkdir /shared
```

This example makes */shared* read-write for the owner and read-only for everyone else:

```
$ chmod 0755 /shared
```

The owner has unrestricted privileges to the directory. Group and world may enter the directory and read files, but not edit or add files.

This example applies the same permissions to the existing contents of the directory, using the *-R* (recursive) option:

```
$ chmod -R 0755 /shared
```

The next example restricts the directory and its existing contents to the directory owner. Files and directories inside the directory may have different owners and permissions, but are still inaccessible to group and world:

```
$ chmod 0700 /shared
```

A common permission set is to give the owner and group the same permissions, such as read-write-execute, and to exclude other:

```
$ chmod 0770 /shared
```

Discussion

You have a lot of power with groups and directories to control file access. Set up groups according to function, for example various teams could each have their own exclusive shared directories. Most shops don't need super-fine-grained control and default to more sharing rather than less. Whatever your needs are, the old *chmod* command is still the fundamental tool for controlling file permissions.

See Also

- *man 1 chmod*

6.8 Using the Special Modes for Special Use Cases

Problem

You want to set some permissions not supported by the traditional user-group-other set of permissions, such as allowing unprivileged users to run a command that requires elevated permissions, protecting files in a directory shared by multiple users, or enforcing certain file permissions in a directory.

Solution

The special modes are *sticky bit*, *setuid*, and *setgid* (see Table 6-3). The sticky bit is applied to directories that contain files owned by multiple users, to prevent users from moving, renaming, or deleting files they do not own:

```
$ chmod -v 1770 /home/duchess/shared
mode of '/home/duchess/shared changed from 0770 (rwxrwx---) to 1770 (rwxrwx--T)
```

setuid is applied to executable files, to elevate any user running the command to the same permissions as the owner:

```
$ chmod 4750 backup-script
mode of 'backup-script' changed from 0750 (rwxrw----) to 4770 (rwsrwx---)
```

Apply *setgid* to a directory, so that all newly created files in the directory are assigned to the same group as the directory's group owner. This is a nice trick for enforcing correct ownership in a shared directory:

```
$ chmod 2770 /home/duchess/shared
mode of '/home/duchess/shared' changed from 0770 (rwxrwx---) to 2770 (rwxrws---)
```

setgid may also be applied to files, changing the effective group of the user to the same group as the file owner.

Discussion

setgid and *setuid* have the potential to create security holes for an intruder or an untrustworthy user. It is a best practice to use them only when you can't think of a safer way to accomplish what you want to do, such as using group assignments or *sudo*.

setuid is useful for executable files.

setgid is useful for directories and files.

The sticky bit is only for directories. Table 6-3 shows the relationship of permissions to owners.

Table 6-3. Octal fields

Mode	Special modes	Owner	Group	World
Read		4	4	4
Write		2	2	2
Execute		1	1	1
setuid	4			
setgid	2			
Sticky bit	1			
No permission	0	0	0	0

The special mode values may be combined (see Table 6-4).

Table 6-4. Sticky bit/setgid/setuid values

Option name	Octal value
No option set	0
Sticky bit set	1
setgid	2
Sticky bit and setgid	3
setuid	4
Sticky bit and setuid	5
setgid and setuid	6
Sticky bit, setgid, and setuid	7

A more descriptive name for the sticky bit is *restricted deletion bit*. This bit prevents unprivileged users from removing or renaming a file in a directory, unless they own the file. You can see this on your */tmp* directory, which is world readable and writeable, and contains files owned by multiple users. Using the sticky bit prevents users from moving, renaming, or deleting files they do not own, even if they have write privileges on some files they do not own:

```
 $ stat --format=%a:%A:%U:%G /tmp
1777:drwxrwxrwt:root:root
```

The sticky bit is the 1 in 1777.

setgid means set group user identification, and *setuid* is set user identification. These are used to elevate the permissions of an unprivileged user to the same as the user or group owner. This is how unprivileged users can use the *passwd* command to change their own passwords, even though only root has write permissions on */etc/passwd*, and everyone else has only read and execute permissions:

```
 $ stat --format=%a:%A:%U:%G /usr/bin/passwd
4755:-rwsr-xr-x:root:root
```

In */etc/passwd* the 4 in 4755 is *setuid*, which means all users have root powers when they run the command, though their powers are limited to changing their own passwords.

See Also

- *man 1 chmod*

6.9 Removing the Special Modes in Octal Notation

Problem

You want to remove the special modes from a file or directory.

Solution

Removing a special mode is a little different from setting it because you need to use an extra leading zero, as in the following example:

```
$ chmod -v 00770 backup.sh
mode of 'backup.sh' changed from 1770 (rwxrwx--T) to 0770 (rwxrwx---)
```

Or replace the leading zeroes with a leading equals sign:

```
$ chmod -v =770 backup.sh
mode of 'backup.sh' changed from 1770 (rwxrwx--T) to 0770 (rwxrwx---)
```

See Also

- *man 1 chmod*

6.10 Setting File Permissions with chmod's Symbolic Notation

Problem

You know that the *chmod* (change mode) command supports both octal and symbolic notation, and you want to use symbolic notation to manage file permissions.

Solution

Symbolic notation is more complex than octal notation and behaves differently according to which operator you use.

There are three operators: +, -, and =. You can change permissions for everyone with the *a* flag, or individually with *u* for the file owner, *g* for the group, and -*o* for other, which is everyone else:

- + adds to existing permissions.
- - subtracts from existing permissions.
- = adds new permissions, and removes any permission bits not listed.

Suppose that *file.txt* is owner read-write, group read, and other read, or *-rw-r--r--*:

```
$ stat --format=%a:%A:%U:%G file.txt
664:-rw-r--r--:stash:stash
```

You want to change it to *-rw-rw-rw-*. Add write permissions to group and other:

```
$ chmod -v g+w,o+w file.txt
mode of 'file.txt' changed from 0644 (rw-r--r--) to 0666 (rw-rw-rw-)
```

You could also use *a=rw*.

In the next example the owner of *file.txt* changes it from world readable and writeable to only the file owner can edit it, and group and world can only read it:

```
$ chmod -v g-w,o-w file.txt
mode of 'file.txt' changed from 0666 (rw-rw-rw-) to 0644 (rw-r--r--)
```

A common permission set is to give the owner and group the same permissions, such as read-write, and to exclude other:

```
$ chmod -v u=rw,g=rw,o-r file.txt
mode of 'file.txt' changed from 0644 (rw-r-r--) to 0660 (rw-rw----)
```

Commands and scripts require the executable bit to be set. This example adds the executable bit to the existing permissions for the file owner:

```
$ chmod -v u+x file.txt
mode of 'file.sh' changed from 0660 (rw-rw----) to 0760 (rwxrw----)
```

The **=** operator is useful for overwriting existing permissions:

```
$ chmod -v u=rw,g=rw,o=r file.txt
mode of 'file.sh' changed from 0760 (rwxrw----) to 0664 (rw-rw-r--)
```

Discussion

The key to using *chmod*'s symbolic notation reliably is to always be explicit and to be mindful of the existing permissions. Add and subtract from existing permissions (except with the = operator, which overwrites), and specify *u*, *g*, *o*, or *-a*.

symbolic notation is designed to be mnemonic, *r* for read, *w* for write, and *x* for execute (Table 6-5).

Table 6-5. Symbolic notation permissions

Mode	Value
r	read
w	write
x	execute

The notation for users and groups is also mnemonic (Table 6-6).

Table 6-6. Symbolic notation owners

Owner	Notation
user	u
group	g
other	o
all	a

Just like octal notation, symbolic notation also supports the special modes (see Recipe 6.11).

There are 10 values in symbolic notation, and unset values (which mean no permissions) are represented by a dash, like this example for Duchess's home directory:

```
$ stat --format=%a:%A:%U:%G /home/duchess
755:drwxr-xr-x:duchess:duchess
```

In *drwxr-xr-x*, the *d* indicates that this is a directory. There is no comparable value in octal notation.

The remaining nine values are divided into three triads, and the three values in each triad represent read, write, and execute.

See Also

- *man 1 chmod*

6.11 Setting the Special Modes with chmod's Symbolic Notation

Problem

You want to set special modes with *chmod*'s symbolic notation.

Solution

The special modes include the *sticky bit*, *setuid*, and *setgid*. These are all set in the executable fields. (See the end of the Discussion in Recipe 6.10 if you are not sure what the executable fields are.)

The sticky bit is applied to directories that contain files owned by multiple users to prevent nonowners from moving, renaming, or deleting the files:

```
$ chmod o+t /shared/stickydir
mode of '/shared/stickydir' changed from 0775 (rwxrwxr-x) to 1775 (rwxrwxr-t)
```

Apply *setgid* to a directory to set all newly created files in the directory to the same group as the directory. This is a nice trick for enforcing correct ownership in a shared directory:

```
$ chmod -v g+s /shared
mode of '/shared' changed from 0770 (rwxrwx---) to 2770 (rwxrws---)
```

Apply *setuid* to an executable file to allow nonroot users to run the executable:

```
$ chmod -v u+s backup-script
mode of 'backup-script' changed from 0755 (rwxr-xr-x) to 4755 (rwsr-xr-x)
```

setuid and *setgid* have the potential to open security holes; see the Discussion to learn more.

Discussion

setuid is useful for executable files.

setgid is useful for directories and files.

The sticky bit is only for directories.

Table 6-7 shows the relationship between owners and modes.

Table 6-7. All symbolic modes

Mode	User	Group	Other
Read	r	r	r
Write	w	w	w
Execute	x	x	x
setuid	s		
setgid		s	
Sticky bit			t

A more descriptive name for the sticky bit is *restricted deletion bit*. This prevents users from removing or renaming a file in a directory unless they own the file. You can see this on your */tmp* directory, which is world readable and writeable, and contains files for multiple users. Using the sticky bit prevents users from moving, renaming, or deleting files they do not own:

```
$ stat --format=%a:%A:%U:%G /tmp
1777:drwxrwxrwt/:root:root
```

setgid means set group identification, and *setuid* is set user identification. These are used to elevate the permissions of an unprivileged user to the same as the file owner. This is how unprivileged users can use the *passwd* command to change their own

passwords, even though only root has write permissions on */etc/passwd*, and everyone else has only read and execute permissions:

```
$ stat --format=%a:%A:%U:%G /usr/bin/passwd
4755:-rwsr-xr-x:root:root
```

rws in the user fields means read, write, and execute for all users, with the same permissions as the file owner.

setgid and *setuid* have the potential to create security holes. It is a best practice to use them only when you can't devise a safer way to accomplish what you want to do, such as using group assignments or *sudo*.

See Also

- *man 1 chmod*

6.12 Setting Permissions in Batches with chmod

Problem

You want to set permissions on more than one file at a time.

Solution

chmod supports operating on lists of files. You can also use the *find* command and shell wildcards to select the files you want to change.



You May Need sudo

If you see "Permission denied" messages, use *sudo*.

The following example takes a space-delimited list of files and makes them all read-only for everyone:

```
$ chmod -v 444 file1 file2 file3
```

Set permissions for a directory and its contents, including subdirectories, with the *-R* (recursive) flag:

```
$ chmod -vR 755 /shared
```

You may use wildcards to select files; for example, to make all *.txt* files in the current directory readable and writable to the owner, and to make group and other readable:

```
$ chmod -v 644 *.txt
```

Use a wildcard to select all filenames that start with the same string:

```
$ chmod -v 644 abcd*
```

This example makes all files in the current directory read-write for the owner and group, without changing permissions on the directory:

```
$ find . -type f -exec chmod -v 660 {} \;
```

You can change the mode of all files belonging to a particular user. You may name the user with either their numeric ID or username. This example starts at the root of the filesystem:

```
$ sudo find / -user madmax -exec chmod -v 660 {} \;
$ sudo find / -user 1007 -exec chmod -v 660 {} \;
```

Discussion

You need root privileges to search for files in all directories.

The dot (*find .*) tells *find* to start its search in the current directory. You can start your search in any directory.

-type limits the results to files, and not directories.

-user looks for files owned by the specified user.

-exec chmod -v 660 {} \; is a fabulous little incantation that takes the results of the *find* search and runs the *chmod -v 660* command on the results. You can use this for pretty much any command that you want to apply to the results of a *find* search.

See Also

- *man 1 chmod*
- *man 1 find*

6.13 Setting File and Directory Ownership with chown

Problem

You need to change ownership on a file or directory.

Solution

Use the *chown* (change owner) command to change file ownership. The basic command syntax is *chown user:group filename*. You may change only the owner, *chown user: filename*, or only the group, *chown :group filename*.

Changing the owner requires root privileges:

```
duchess@client1:~$ sudo chown -v madmax: song.wav
changed ownership of 'song.wav' from duchess:duchess to madmax:duchess
```

Change the group owner:

```
$ sudo chown -v :composers song.wav
changed ownership of 'song.wav' from madmax:duchess to :composers
```

Change both the user and group owner:

```
$ sudo chown stash:stash song.wav
```

Discussion

You need root privileges to make changes to files you do not own and to transfer file ownership to another user. You can change group file ownership without root privileges when you belong to both the original group and the new group.

The colon is optional when you change only the owner and required when you change the group.

See Also

- *man 1 chown*

6.14 Changing Ownership on Batches of Files with chown

Problem

You want to change ownership of directories and their contents, or just the contents of directories, a list of files, or change ownership of files from one user to another.

Solution

chown supports operating on lists of files. You can also use the *find* command and shell wildcards to list the files you want to change.

To change the owner of several files at once with *chown*, use a space-delimited list:

```
$ sudo chown -v madmax:share file1 file2 file3
```

Change files with a certain file extension in the current directory to a new group:

```
$ sudo chown -v :share *.txt
```

Give all of a user's files in a directory to another user, using their numeric UIDs or usernames:

```
$ chown -Rv --from duchess stash /shared/compositions

$ chown -Rv --from 1001 1005 /shared/compositions
```

Use the -find_ command to traverse the entire filesystem, or any directory and its subdirectories, to give all of a user's files to another user:

```
$ sudo find / -user duchess -exec chown -v stash {} \;

$ sudo find / -user 1001 -exec chown -v 1005 {} \;
```

Discussion

Transferring ownership of all of a user's files to another user, or to a different group, is useful for cleaning up after users who no longer have accounts on the system.

See Also

- *man 1 chown*

6.15 Setting Default Permissions with umask

Problem

You want to understand why files are created with a certain set of default permissions, and how to configure the defaults yourself.

Solution

The umask (user file-creation mode mask) controls this behavior. To see what yours is, run the *umask* command:

```
$ umask
0002
```

This is how it looks it in symbolic notation:

```
$ umask -S
u=rwx,g=rwx,o=rx
```

This sets your default permissions to 0775 for directories and 0664 for files, because the umask "masks" the hardcoded default permissions of 0777 and 0666. Or you can think of it as subtraction, 0777 - 0002 = 0775.

To change your umask temporarily for the duration of your current session, set it this way:

```
$ umask 0022
```

Set the umask permanently by inserting the line *umask 0022*, or whatever value you want, in your *~/.bashrc* file.

Set the default umask for all of your users in */etc/login.defs*:

```
UMASK 022
```

Table 6-8 shows some common umask values.

Discussion

umask is a Bash shell built-in, and not an executable program stored in */bin*, */usr/bin*, or any of the other *bin* (binary) directories.

Table 6-8 lists some commonly used umask values.

Table 6-8. Common umask values

umask	Directories	Files
0002	0775	0664
0022	0755	0644
0007	0770	0660
0077	0700	0600

See Also

- *man 1 chmod*
- See the Shell Builtin Commands section of *man 1 bash* to learn more about *umask* and other Bash built-in commands

6.16 Creating Shortcuts (Soft and Hard Links) to Files and Directories

Problem

You want to create shortcuts or links to files.

Solution

There are two types of links in Linux: soft links and hard links. Soft links are for files and directories. Hard links are only for files.

Use the *ln* (link) command to create soft and hard links. The following example creates a soft link to an external directory, */files/userstuff*, in Mad Max's home directory:

```
$ ln -s /files/userstuff stuff
```

/files/userstuff is the target, and *stuff* is the destination, or soft link name. You can name your soft links anything you want, and move and delete them without affecting their targets. When you open a soft link, it behaves the same way as opening the target.

Hard links are copies of files. The default for the *ln* command is to create hard links:

```
$ ln /files/config1.txt myconf.txt
```

Discussion

Soft links are for files and directories, while hard links are only for files.

Soft links

Soft links are more commonly called *symlinks*, short for symbolic links.

Symlinks point to files and directories. When the target of a symlink is deleted, renamed, or moved, the symlink is broken. If you create a new file with the same name as the deleted file, the symlink is restored, even if the content is different.

Symlinks can cross filesystems. You can even create symlinks to files or directories that are not permanently available, like USB storage devices or network file shares.

Symlinks are not updated when the target changes (renamed, moved, or deleted). You need to create a new symlink and delete the old one.

You don't manage permissions or ownership on symlinks because only the permissions on the target matter.

Symlinks look like this:

```
$ stat stuff
  File: stuff -> /files/userstuff
  Size: 4               Blocks: 0          IO Block: 4096   symbolic link
Device: 804h/2052d      Inode: 877581      Links: 1
Access: (0777/lrwxrwxrwx)  Uid: ( 1000/ madmax) Gid: ( 1000/ madmax)
```

File: stuff → /files/userstuff shows the target that the symlink points to.

The third line identifies this as a symbolic link.

The *l* in *Access: lrwxrwxrwx* identifies this as a symlink.

This is what a symlink looks like in a file listing:

```
$ ls -l
[...]
lrwxrwxrwx 1 madmax madmax  4 Apr 26 12:42 stuff -> /files/userstuff
```

Hard links

Files are uniquely identified by *inodes*, and inodes are what hard links point to, rather than filenames. The *ls* command shows inodes with the *-i* option. The inode in this example is 1353, and it is the same for the three hard links:

```
$ ls -li
1353 -rw-rw-r--   3 madmax madmax  11208 Apr 26 13:06  config.txt
1353 -rw-rw-r--   3 madmax madmax  11208 Apr 26 13:06  config2.txt
1353 -rw-rw-r--   3 madmax madmax  11208 Apr 26 13:06  config3.txt
```

This is because all three inodes point to the same block of data.

Hard links always work because they point directly to inodes. Files with multiple hard links can be moved, renamed, and edited, and all hard links remain in sync because they all point to the same data block.

Every file on a Linux system starts with a hard link. When you create a hard link, you are creating a new filename for an existing data block.

Hard links cannot cross filesystems, but exist only inside a single filesystem. For example, if you have / and */home* on separate partitions, you cannot make hard links in */home* for files in /.

You can make as many hard links to a file as you like, and the disk space occupied by the data they point to is always the same, regardless of how many hard links it has.

Contrast hard links with making file copies: every copy uses more disk space, each copy is independent, and copies can go anywhere.

A file is not completely deleted until all hard links are deleted. You can see this with *ls*. The following example shows another view of our example inode with three hard links:

```
$ stat config3.txt
  File: config3.txt
  Size: 11208           Blocks: 24         IO Block: 4096   regular file
Device: 804h/2052d      Inode: 1353        Links: 3
```

Compare the File, Size, and Links to a symlink. A hard link is a regular file, and note Links: 3. This shows there are three hard links to the same data. When you delete a file with more than one hard link, it is not deleted until you delete all of them. Locate all related hard links with the *find* command:

```
$ find /etc -xdev -samefile config3.txt
./config
./config2
./config3
```

Symlinks are used a lot in Linux, hard links not so much. Some backup applications use inodes for deduplication. In olden times, when filesystems were much smaller,

running out of inodes was not uncommon. In this case hard links were preferable, because symlinks each have their own inodes, but hard links share inodes.

You can see how many inodes a filesystem has with the *du* command, and how many are used:

```
$ df -i /dev/sda4
Filesystem        Inodes  IUsed     IFree IUse% Mounted on
/dev/sda4      384061120 389965 383671155    1% /home
```

With 1% in use, I'm not running out of inodes anytime soon.

See Also

- man 1 ls

6.17 Hiding Files and Directories

Problem

You want to hide some files and directories so that nobody can see them.

Solution

To hide files so nobody can see them, put them on a storage device only you have access to.

To reduce clutter in your file manager, use *dot files* to ignore files. You already have these. Look for a setting like "Show hidden files" in your graphical file manager, or use the *-a* option for *ls*:

```
$ ls -a
.
..
Audiobooks
.bash_history
.bash_logout
.bashrc
bin
.bogofilter
.cache
Calibre-Library
cat-memes
.cddb
.cert
```

Prefixing any file with a dot makes it a hidden file, though it is really not hidden, but ignored until you want to see it. This is used mainly in users' home directories to

reduce clutter by not displaying configuration files. These are normal files you can edit, delete, or whatever you want.

Discussion

Note the single and double dots at the top of the file list. The single dot represents the current directory, and the double dot represents the parent directory. Try it with the *cd* command. The first example stays in the current directory, the second example changes to the parent directory:

```
stash@client4:~$ cd .
stash@client4:~$

stash@client4:~$ cd ..
stash@client4:/home$
```

Run *cd* with no options to return to your home directory, or *cd -* to return to the last directory you were in.

See Also

- See the Shell Builtin Commands section of *man 1 bash* to learn more about *cd* and other Bash built-in commands

CHAPTER 7

Backup and Recovery with rsync and cp

You know you need to make good backups of your computer files and to test them periodically to see if you can restore your files. But how do you do this on Linux? Fear not, for backups and restores on Linux are quite understandable, and your backup files are easy to search and restore.

It is helpful to have a couple of USB sticks for practicing the commands in this chapter and a few directories full of files that you won't mind losing, should anything go wrong.

We will use *rsync* and *cp*. Both are essential Linux tools, and you can count on them being well maintained and available.

cp is the copy command included in the GNU *coreutils* package, which is installed by default on nearly every Linux distribution. *cp* is for simple copying. It may be all you need to maintain regular backups.

rsync is an efficient file-transfer program, and its main purpose is keeping filesystems in sync with each other. When you use it for making backups, it keeps your local files in sync with your backup device. It is fast and efficient because it transfers only the changes in files. Unlike a lot of backup software, which never want you to delete anything, it even mirrors deletions. Because of these features, rsync is the tool of choice for updating and mirroring user home directories, websites, git repositories, and other large complex file trees.

There are two ways to use rsync over a network: over SSH, for authenticated login and transport, or by running it as a daemon. Using SSH requires users to have login accounts on every machine for which they need rsync access. When rsync is run in daemon mode, you can use its built-in authentication methods to control access so that users do not need login accounts on the rsync server. Daemon mode is

well-suited for a LAN backup server. It is not safe to access over untrusted networks, unless you use a VPN (see Chapter 13).

What sort of device do you store your backups on? This depends on your needs. I am a fan of USB storage media for a single user. Suppose you have a desktop Linux PC, a laptop, a tablet, and a smartphone. Back up your phone and tablet to your PC, then back up the PC to a USB hard drive. Super-important files could go to an online backup service.

For multiple users, a good solution is a central backup server. This can be any Linux PC.

Consider longevity. You cannot count on longevity with digital storage media, because even if the medium (hard disk, USB storage drive, CD/DVD) survives, there is no guarantee that the tools to read it will endure. Hardware and file formats change. Can you still read floppy disks? Remember Zip disks? How about those archives of old Microsoft Word and Powerpoint documents? With open source file formats you can always find a way to recover them. Good luck with proprietary formats when the vendor decides to stop supporting them.

Paper is still the long-term storage champion, and worth considering for your most important documents and photos.

For long-term digital storage, plan to transfer your archives to new media periodically, possibly with new commands and to new filesystem formats.

What about backing up your backup server? No problem. Setting up a remote rsync mirror for backing up the backups is a common strategy, if your internet connection is robust enough to handle the traffic. But before you build a massive backup infrastructure, think about how many levels of redundancy you really need. Offsite backups are insurance against a disaster at your site. This can be a remote backup server at a site you control, or a friend's site, or rented space in a datacenter. Maybe regular drops of an external hard drive to a bank safe-deposit box would be enough for your needs. Also think about recovery: can you get to your backups quickly?

Always remember that the purpose of backups is *recovery*. Test your backups regularly to avoid learning the hard way that your backup method failed.

In this chapter you will learn about simple copying to USB storage devices with *cp*. For some users this is all they will ever need.

Most of the chapter is about using the *rsync* command for faster and more efficient copying. You can use *rsync* to back up your files to local media or remote servers. You will learn which files you should back up, how to fine-tune your file selection, maintain the same file permissions and timestamps on your files, how to build an rsync backup server for multiple users, and how to make secure remote backups.

7.1 Selecting Which Files to Back Up

Problem

You're not sure which files you should back up. Do you need to make backups of your system files? Do you really need to back up all of your personal files? Are there files you should not back up?

Solution

Any file that you would be sorry to lose is a file you need to back up. Your personal files and system data files are the most important. Restoring system files such as commands, applications, and libraries is less important because you can always download and reinstall these.

The following directories contain files such as configurations; data files for servers such as web, FTP, and mail servers; log files; applications installed in nonstandard locations; and shared directories, all of which should be backed up:

- */boot/grub*, if it contains any customizations such as themes, background images, or fonts.
- */etc* contains system configuration files.
- */home*, users' personal files.
- */mnt*, temporary filesystem mountpoints. Back this up if you have mountpoints you want to preserve.
- */opt*, for proprietary or other applications not installed the standard way.
- */root*, the root user's personal files.
- */srv*, data for servers such as web, FTP, and rsync servers.
- */tmp* holds temporary data that is automatically updated or deleted as needed. Some of the data in */tmp* is persistent, for example user-created files and some system services, and they should be backed up.
- */var* stores many types of data such as log files, mail spools, cron jobs, and data for system services, though most distros have migrated to using */srv* for system services.

If you have any shared directories, custom commands and scripts, or any data files or directories not listed previously, back them up.

/proc, */sys*, and */dev* are pseudofilesystems that exist only in memory and should not be backed up.

/media is for mounting removable storage media and should be managed by the system, so there is no need to back it up. If you are manually creating mountpoints in */media*, they really need to be moved to */mnt*.

Many databases should not be backed up with simple copying because they have special utilities and procedures for making copies and backups, and for restoring from backups. Use the tools made for your databases. Some examples are PostrgreSQL, MariaDB, and MySQL.



Restoring From Backup

Some files should not be restored from backup; see Recipe 7.2.

Copying everything is the easy way if your backup storage is large enough. You also have the option of fine-tuning your file selection by creating lists of files to copy or to exclude; see Recipes 7.8 and 7.9.

Discussion

Storage media is so cheap now you may not have to care about conserving storage space. If you need to be mindful of storage limitations, see the recipes in this chapter on file selection.

See Also

- Filesystem Hierarchy Standard

7.2 Selecting Files to Restore from Backups

Problem

You are restoring files from backup, and you want to know if there are files that should not be restored.

Solution

Some files should not be restored, depending on the circumstances.

Do not restore */etc/fstab* after reinstalling Linux (the file that configures your static filesystem mounts). Every time you install Linux, all the filesystems get new Universal Unique Identifiers (UUIDs), so they will not be recognized and your new installation will fail.

Be careful restoring any file in */etc* or the dotfiles (such as */home/.config* or */home/.local*) in your home directory. If you are restoring from backup to a new installation of a different release, or a different Linux distribution, there may be incompatibilities in configuration options or file locations. Restore them one at a time so you can quickly spot any problems.

See Also

- Chapter 1

7.3 Using the Simplest Local Backup Method

Problem

You want to know the easiest, simplest way to make regular backups to a local USB storage device.

Solution

The answer is to use simple copying. Get yourself a nice USB hard drive or USB stick. Plug it in and use your file manager to copy your files. Easy peasey, no muss, no fuss, and it is dead simple to restore your files. Or, use the *cp* command (see Recipe 7.4).

Discussion

Simple copying doesn't scale all that well, but for a few devices, like a PC, laptop, and phone, it works fine. The important part is making regular backups, verifying that you can restore files from your backups, and not worrying if you're being nerdy enough.

Back in olden times, backups were more complicated because storage was expensive, so backup programs used a lot of tricks to save space. Now you can buy multi-terabyte external USB 3.0 hard drives for less than $200.

See Also

- *man 1 cp*

7.4 Automating Simple Local Backups

Problem

You like using simple copying for making your backups to an external USB storage drive, and you want to automate the process.

Solution

This calls for the *cp* command and *crontab* to schedule your backups.

You may list individual files and directories to copy with *cp*, separated by spaces:

```
duchess@pc:~$ cp -auv Pictures/cat-desk.jpg Pictures/cat-chair.png \
  ~/cat-pics /media/duchess/2tbdisk/backups/
```

The following example copies Duchess's entire home directory to the *backups* directory on an external USB drive named *2tbdisk*:

```
duchess@pc:~$ cp -auv ~ /media/duchess/2tbdisk/backups/
```

This creates */media/duchess/2tbdisk/backups/duchess/* on the backup device.

Copy the contents of a directory without copying the directory itself:

```
duchess@pc:~$ cp -auv /home/duchess/* /media/duchess/2tbdisk/backups/
```

Create a personal cron job to run your backup every night at 10:30 P.M.:

```
duchess@pc:~$ crontab -e
# m h  dom mon dow   command
30 22  *   *   *   /bin/cp -au /home/duchess /media/duchess/2tbdisk/backups/
```

Discussion

If you want to preserve file attributes such as ownership and permissions, format your backup drive with a Linux filesystem that supports file attributes, such as Ext4, XFS, or Btrfs (see Chapter 11). The FAT filesystems do not preserve ownership or permissions.

Keep an eye on how long it takes your backup to run. If it takes longer than your scheduled backup interval, cron will start the next backup on schedule, and then you have a mess.

The first run takes the longest because all the files are new. Subsequent backups will go faster as only new files and files with newer timestamps will be copied.

The tilde, ~, is a shortcut for the current user's home directory, so in this recipe it is short for */home/duchess*.

The asterisk in */home/duchess/** means copy all the files in */home/duchess*, but not the directory */home/duchess*.

The *-a*, *-u*, and *-v* options for *cp* mean:

- *-a, --archive* recursively copies and preserves all file attributes: mode, ownership, timestamps, and extended attributes.
- *-u, --update* tells *cp* to copy only files with newer timestamps than the copies in the backup directory, or new files that have not been backed up yet.
- *-v, --verbose* prints activity messages during the copy operation.

Some other useful options:

- *-R, -r* recursive; use this to copy directories when you do not use the *-a* option. *-a* preserves file attributes, *-R, -r* does not. FAT and exFAT filesystems do not support file attributes, so use *-R, -r* with these.
- *--parents* creates missing parent directories on the destination.
- *-x, --one-file-system* This prevents recursing into other partitions and mounted network filesystems. For example, if you have an NFS share mounted you may not want to add it to your backup.

Most Linux distributions mount USB devices in */run/media* or */media*. The easy way to find the filepath for your USB drive is to look in your file manager or use the *lsblk* command:

```
$ lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
[...]
sdb      8:16   0  1.8T  0 disk
└─sdb1   8:17   0  1.5T  0 part /media/duchess/backups
```

See Also

- Recipe 3.7 to learn more about using *cron*
- *man 1 crontab*
- *man 1 cp*

7.5 Using rsync for Local Backups

Problem

You want to make backups to a USB stick or USB hard drive, and you want something that is faster and more efficient than simple copying. You also want simple file restorations that you can make with standard Linux tools, without needing special software.

Solution

rsync is what you want. It keeps filesystems synchronized, both local and remote. *rsync* is fast and efficient, as it transfers only the changes in files, and you can restore files with the *rsync* command, the *cp* command, your file manager, or whatever copying tool you prefer.

The following example shows how to back up a home directory. First, name your source directory, which is the directory you want to back up, then name the destination directory. This example copies Duchess's */home* to a USB drive named *2tbdisk*:

```
duchess@pc:~$ rsync -av ~ /media/duchess/2tbdisk/
sending incremental file list
duchess/
duchess/Documents/
duchess/Downloads/
duchess/Music/
[...]

sent 27,708,209 bytes  received 20,948 bytes  11,091,662.80 bytes/sec
total size is 785,103,770,793  speedup is 28,313.29
```

You can specify two or more directories, in a space-delimited list, to transfer to the destination directory:

```
duchess@pc:~$ rsync -av ~/arias ~/overtures /media/duchess/2tbdisk/duchess/
```

Copy files from your backup device to your computer by reversing the source and destination:

```
duchess@pc:~$ rsync -av /media/duchess/2tbdisk/duchess/arias /home/duchess/
```

You may safely test your *rsync* command, without copying any files, with the *--dry-run* option:

```
duchess@pc:~$ rsync -av --dry-run \
~/Music/scores ~/Music/woodwinds /media/duchess/2tbdisk/duchess/
```

If any files are deleted from a source directory, *rsync* will not delete them from the destination directory unless you explicitly tell it to with the *delete* option:

```
duchess@pc:~$ rsync -av --delete /home/duchess /media/duchess/2tbdisk/
```

Discussion

The tilde, ~, is a shortcut for your home directory, so in the examples it means */home/duchess*.

Command examples with line breaks use a slash, \, to indicate that the command continues on the next line. You can copy the whole command, with the slashes, and it should work.

If you have networked filesystems mounted on your PC, such as NFS or Samba, use the *-x* option to copy only from your local filesystem, and not recurse into the remote filesystems.

Adding a trailing slash, ~/, (*/home/duchess/*) copies only the contents of the *duchess/* directory, but not the directory itself, resulting in */media/duchess/2tbdisk/[files]*. Omitting the trailing slash transfers the contents of */home/duchess* and the *duchess* directory, resulting in */media/duchess/2tbdisk/duchess/[files]*. The trailing slash only matters on the source directory, and it makes no difference on the destination directory.

Don't feel bad if you have to count on your fingers or make a lot of test runs to remind yourself how the trailing slash behaves, because this vexes everyone. It might help to think of the trailing slash as a little fence that prevents the source directory from escaping.

The *-a* and *-v* options for *rsync* mean:

- *-a, --archive* retains the mode, timestamps, permissions, and ownership, and copies recursively. This is the same as **-rlptgoD**, which copies recursively, copies symlinks, preserves permissions, preserves file modification times, preserves file ownership, and preserves special files, such as device files.
- *-v, --verbose* displays activity messages.

You may wish to use some of these options:

- *-q, --quiet* suppresses nonerror messages.
- *--progress* shows information on each file as it is transferred.
- *-A, --als* preserves access control lists (ACLs).
- *-X, --xattrs* preserves extended file attributes (xattrs).

Your filenames, of course, will be different than the examples. *2tbdisk* is a filesystem label created by the user (see Recipe 9.4). It is an abbreviation for "2 terabyte disk." If you do not create a label, *udev* creates one, for example */media/duchess/488B-7971/*.

You may use the normal Linux tools, such as *rsync*, your file manager, or the *cp* command, to restore files.

See Also

- *man 1 rsync*

7.6 Making Secure Remote File Transfers with rsync over SSH

Problem

You want to use *rsync* to copy files to another computer on your local network or over the internet, and you want encrypted transport and authentication.

Solution

rsync uses SSH by default when you transfer files to another machine. The remote machine must be running an SSH server, and the source machine must have an SSH client already set up (see Chapter 12).

This example transfers files over the local network from Duchess's PC to her laptop. Duchess's username on her laptop is Empress, and she is copying files from her home directory on her PC to her home directory on her laptop:

```
duchess@pc:~$ rsync -av ~/Music/arias empress@laptop:songs/
duchess@laptop's password:
building file list ... done
arias/
arias/o-mio-babbino-caro.ogg
arias/deh-vieni-non-tardar.ogg
arias/mi-chiamano-mimi.ogg
wrote 25984 bytes  read 68 bytes  7443.43 bytes/sec
total size is 25666  speedup is 0.99
```

If the destination directory does not exist, *rsync* will create it.

To upload files over the internet, use the fully qualified domain name of the server you are logging in to:

```
duchess@pc:~$ rsync -av ~/Music/woodwinds \
 empress@remote.example.com:/backups/
```

The syntax for copying files from a remote host is reversed. This example copies the */woodwinds* directory and its contents from the remote host to Duchess's home directory:

```
duchess@pc:~$ rsync -av empress@remote.example.com:/backups/woodwinds \
 /home/duchess/Music/
```

Discussion

You might remember when the SSH option had to be explicit—for example, *rsync -a -e ssh [options]*. This is no longer necessary.

You may find some of these options to be useful:

- *--partial* preserves partially downloaded files when the network connection is interrupted, and resumes the file transfer from where it left off when the connection is restored.
- *-h, --human-readable* displays file sizes in kilobytes, megabytes, and gigabytes, rather than bytes.
- *--log-file=* stores a complete record of each transfer in a text file. Put them all together like this:

```
duchess@pc:~$ rsync --partial --progress \
 --log-file=/home/duchess/rsynclog.txt \
 -hav ~/Music/arias empress@remote.example.com:/backups/
```

Both authentication and transport are encrypted by SSH. Users need shell accounts on all machines they are going to transfer files to. See Chapter 12 to learn about secure remote administration with SSH.

Consider setting up a central backup server for simplifying administration. Your users have their own accounts with their own */home* directories and can manage their own backups and restores without bothering you.

Another option for a backup server is to run *rsync* as a service. The advantage of this is your *rsync* users do not need login accounts on the server. One disadvantage is it does not support encrypted transfers. See Recipe 7.13 to learn about this.

See Also

- *man 1 rsync*
- Recipe 12.5
- Recipe 12.7

7.7 Automating rsync Transfers with cron and SSH

Problem

You want to create crontabs to automatically run your secure *rsync* transfers.

Solution

You need SSH set up for passwordless authentication on the destination machine (see Recipes 12.10 and 12.11) and network access to the destination machine for the clients.

Then use */etc/crontab* for transfers that require root permissions. The following example makes a backup of */etc* every night at 10 P.M. to a LAN server named *server1*:

```
# m h dom mon dow user  command
00 22 * * * root /usr/bin/rsync -a /etc server1:/system-backups
```

Use personal crontabs for transferring your own files (see Recipe 3.7).

Discussion

OpenSSH is a lovely tool that provides secure network transfers for a host of tasks. Anything that you run over a network can probably run over SSH.

See Also

- Chapter 12
- Recipe 12.10
- Recipe 12.11

7.8 Excluding Files from Backup

Problem

So far the examples have shown how to transfer entire directories. You want to know how to exclude files and directories from being copied.

Solution

For simplicity, the following examples demonstrate local transfers to a USB drive, but they also work for remote transfers over SSH; see Recipe 7.6.

When it's just a few files, you can list them on the command line using *--exclude=*. This example excludes one file from */home/duchess/Music/arias*:

```
duchess@pc:~$ rsync -av --exclude=lho-perduta.wav \
 ~/Music/arias /media/duchess/2tbdisk/duchess/Music/
```

This is nice and easy and reliable. However, there is a gotcha: if there are multiple files in your source directory with the same name as your excluded file, all of them will be excluded. If you do not want the duplicates to be excluded, you need to specify which one is to be excluded. In the following example you want to exclude only the copy in the *arias/* source directory:

```
duchess@pc:~$ rsync -av --exclude=arias/lho-perduta.wav \
 ~/Music/arias /media/duchess/2tbdisk/duchess/Music/
```

Exclude more than one file by enclosing them in curly braces, separated with single quotes and commas. There must be no spaces between the equals sign and the curly brace, and no spaces between the commas and single quotes:

```
duchess@pc:~$ rsync -av \
--exclude={'arias/lho-perduta.wav','non-mi-dir.wav','un-bel-di-vedremo.flac'} \
~/Music/arias /media/duchess/2tbdisk/duchess/Music/
```

Excluding directories works the same way as excluding files, and you can mix files and directories in your exclude list:

```
duchess@pc:~$ rsync -av \
--exclude={'soprano/','tenor/','non-mi-dir.wav'} \
~/Music/arias /media/duchess/2tbdisk/duchess/Music/
```

See Recipe 7.11 to learn how to put your excludes list in a file.

Discussion

The root directory in an *rsync* transfer is the top-level directory you are transferring files from. In the examples in this recipe, that is *~/Music/arias*. *rsync* checks all the files and directories in your root directory, and compares them to your exclude directives, which *rsync* calls *patterns*. Patterns are checked against the files and directories in your root directory, starting at the root and proceeding down through the directory hierarchy. Every time a pattern is matched, it is excluded from the transfer. If the pattern *arias/lho-perduta.wav* is duplicated in another location, such as *2arias/lho-perduta.wav*, it will also be excluded. When a pattern ends with a slash (*/*), *rsync* will match only directories.

See Also

- *man 1 rsync*
- Recipe 7.10

7.9 Including Selected Files to Backup

Problem

You want to include a selected set of files in your backup, rather than defining a list of files to exclude.

Solution

When you want to back up just a few files, you can do this on the command line. *--include=* operates differently than *--exclude=* because it doesn't really mean "include," it means "do not exclude." It needs two additional options, *--include=*/* and *--exclude='*'*, as this example of transferring a single file shows:

```
duchess@pc:~$ rsync -av --include=*/ --include=lho-perduta.wav \
 --exclude='*' ~/Music/arias /media/duchess/2tbdisk/duchess/Music/
```

You may transfer a list of files:

```
duchess@pc:~$ rsync -av --include=*/ \
--include={'lho-perduta.wav','non-mi-dir.wav','un-bel-di-vedremo.flac'} \
--exclude='*' ~/Music/arias /media/duchess/2tbdisk/duchess/Music/
```

There must be no spaces between the equals sign and the curly brace, and no spaces between the commas and single quotes.

If there are multiple files with the same name in different locations in your source directory, *rsync* will transfer all of them. In this example only */home/duchess/Music/arias/sopranos/lho-perduta.wav* is transferred because the pattern *soprano/lho-perduta.wav* is unique in */Music/arias*:

```
duchess@pc:~$ rsync -av --include=*/ --include=soprano/lho-perduta.wav
--exclude='*' ~/Music/arias /media/duchess/2tbdisk/duchess/Music/
Music/
Music/arias/
Music/arias/baritone/
Music/arias/soprano/
Music/arias/soprano/lho-perduta.wav
Music/arias/tenor/
[...]
```

That transfers only a single file, but all the subdirectories in *~/Music/arias* are copied. Use the *-m, --prune-empty-dirs* option to prevent copying empty directories, like this example:

```
duchess@pc:~$ rsync -avm --include=*/ --include=soprano/lho-perduta.wav
--exclude='*' ~/Music/arias /media/duchess/2tbdisk/duchess/Music/
Music/
Music/arias/soprano/
Music/arias/soprano/lho-perduta.wav
```

When you have more than a few files to include, store your list in a plain-text file (see Recipes 7.10 and 7.11).

Discussion

--include=/* tells *rsync* to traverse your entire source directory.

--include=[files] means do not exclude these files.

--exclude=''* tells *rsync* to exclude everything not included.

Remember that all filepaths are relative to your source directory and not your system's root directory.

See Also

- *man 1 rsync*
- Recipe 7.10
- Recipe 7.11

7.10 Managing Includes with a Simple Include File

Problem

Your includes are too many for a command-line incantation, and you want to maintain your list in a file that *rsync* can read. You also want this to be simple, if possible, thanks to past experience with *rsync* include/exclude files that never would work right.

Solution

The simplest way to maintain a list, without going crazy over figuring out *rsync*'s include/exclude syntax, is to create a plain list of files that you provide to the *--files-from=* option. You don't have to worry about getting them in the right order, or using *rsync*'s filter notation, just a plain list with whatever files and directories you want. The only gotcha is every item in your list must be relative to your source directory. In the following example, all list items are relative to */home/duchess*:

```
# include file list
#
/Documents/compositions/jazz/
/Documents/schedule.odt
/Videos/concerts/
.config
.local
```

```
/Music/courses/bassoon.avi</strong>
[...]
```

Then use the list with the *--files-from* option:

```
duchess@pc:~$ rsync -av ~ --files-from ~/include-list.txt \
 duchess@remote.example.com:/backups/
```

Discussion

This is the easiest way to maintain a list of files and directories to back up. There are no excludes, no wildcards, no funny syntax, just a nice clean understandable list.

When you use the tilde to indicate your home directory, leave off the equals sign from *--files-from*, like the last example in the recipe.

See Also

- *man 1 rsync*
- Recipe 7.9
- Recipe 7.11

7.11 Managing Includes and Excludes with an Exclude File

Problem

You like the simple include file concept in Recipe 7.10, but you really want to have both includes and excludes.

Solution

What you want is an *rsync* exclude file. An exclude file provides more flexibility and contains both includes and excludes. The following example illustrates a basic configuration. Every item must start by including the source root, which in this example is */home/duchess*, and end with excluding the source root:

```
# exclude file list
#
# include home directory
+ /duchess/
#
# include .config and .local, exclude all other dotfiles
+ /duchess/.config
+ /duchess/.local
- /duchess/.*
#
# include jazz/, exclude all other files in Documents
```

```
+ /duchess/Documents/
+ /duchess/Documents/compositions/
+ /duchess/Documents/compositions/jazz/
- /duchess/Documents/compositions/*
- /duchess/Documents/*
#
# include schedule.odt, include all .ogg files in
# arias/, exclude all other files in Music
+ /duchess/Music/
+ /duchess/Music/schedule.odt
+ /duchess/Music/arias/*.ogg
- /duchess/Music/arias/*
- /duchess/Music/*
#
# includes courses/, exclude all other files in Videos
+ /duchess/Videos/
+ /duchess/Videos/courses/
- /duchess/Videos/*
#
# exclude everything else
- /duchess/*
```

Feed it to *rsync* with the *exclude-from=* option:

```
duchess@pc:~$ rsync -av ~ \
 --exclude-from=/home/duchess/exclude-list.txt \
 /media/duchess/2tbdisk/
```

Discussion

The *exclude-list.txt* example demonstrates backing up:

- Two dotfiles, *.config* and *.local*
- A single subdirectory of */Documents*, */jazz*
- A single file in */Music*, *schedule.odt*, and only *.ogg* files in */Music/arias/*
- A single directory in */Videos*, */courses*

There must be no spaces between lines, and comments (#) are useful for reminding you of the purpose of each section, and for adding a bit of whitespace. Preface your includes with the plus sign and the excludes with the minus sign.

All other files in */home/duchess* are excluded from backup. Includes must always come first. As the example file shows, you have to be precise in defining each include/exclude. Includes must be listed in their directory hierarchy order, with all subdirectories. For example, this will fail with all files excluded:

```
+ /duchess/Documents/compositions/
- /duchess/*
```

Try including */Documents*:

```
+ /duchess/Documents/
+ /duchess/Documents/compositions/
- /duchess/*
```

Now all the subdirectories and their contents in */Documents* are transferred, and not just */compositions*. To copy only */compositions* you have to exclude */Documents*; it is not enough to exclude only */duchess*. The following example copies only */duchess/Documents/compositions/* and nothing else:

```
+ /duchess/Documents/
+ /duchess/Documents/compositions/
- /duchess/Documents/*
- /duchess/*
```

You may use wildcards to include or exclude files by type. For example, include all *.ogg* and *.flac* files, exclude all *.wav* files, and exclude all *cache* and *temp* directories:

```
# include home directory
+ /duchess/
#
# include all ogg and flac files
+ *.ogg
+ *.flac
#
# exclude wav files, all cache and temp dirs
- *.wav
- cache*
- temp*
```

You may have multiple source directories.

There is always just one destination directory.

See also

- *man 1 rsync*
- Recipe 7.10

7.12 Limiting rsync's Bandwidth Use

Problem

Large file transfers can use a lot of network bandwidth and slow down everything. You want a simple way to restrict *rsync*'s bandwidth use without implementing something complex like traffic shaping.

Solution

Use *rsync*'s *--bwlimit* option. This example limits it to 512 Kbps:

```
$ rsync --bwlimit=512 -ave ssh ~/Music/arias empress@laptop:songs/
```

Discussion

--bwlimit only accepts values in kilobits.

See Also

- *man 1 rsync*

7.13 Building an rsyncd Backup Server

Problem

You want your users to back up their own data on a central backup server, but you don't want to give them shell accounts on your backup server.

Solution

Set up a central backup server, and run *rsync* in daemon mode. You should have name services already set up, and the hosts on your network have access to the backup server. Users will not need login accounts on the server because you will use *rsync*'s own access controls and user authorization to control access to the *rsync* archives.



For LAN Use Only

This is suitable for LAN use only, and not over untrusted networks, because the *rsync* daemon does not encrypt the authentication or file transfers. For encrypted transfers you need OpenVPN (Chapter 13).

rsync must be installed on all machines. *rsyncd* runs on the backup server, and clients will use the *rsync* command to connect to the server.

On the backup server, edit or create */etc/rsyncd.conf* to create an *rsync* module defining the archive:

```
# modules
[backup_dir1]
   path = /backups
   comment = "server1 public archive"
```

```
list = yes
read only = no
use chroot = no
uid = 0
gid = 0
```

Create your */backups* directory, mode 0700, owned by root, to prevent unauthorized access from anyone who has access to the server:

```
$ sudo mkdir /backups/
$ sudo chmod 0700 /backups/
```

Start *rsyncd* on the server in daemon mode with systemd:

```
$ sudo systemctl start rsyncd.service
```

On Debian/Ubuntu it is *rsync.service*.

If your Linux does not have systemd, start it with the *rsync* command:

```
admin@server1:~$ sudo rsync --daemon
```

On the backup server, test that *rsyncd* is listening and accepting connections:

```
admin@server1:~$ rsync server1::
backup_dir1     "server1 public archive"
```

Then test from another PC on your network using the server's hostname or IP address:

```
duchess@pc:~$ rsync server1::
backup_dir1     "server1 public archive"

duchess@pc:~$ rsync 192.168.10.15::
backup_dir1     "server1 public archive"
```

Now you know that it is ready to transfer files. Test that you can copy files to your new *rsyncd* server:

```
duchess@pc:~$ rsync -av ~/drawings server1::backup_dir1
building file list.....done
drawings/
drawings/aug_03
drawings/sept_03

wrote 1126399 bytes  read 104 bytes  1522.0 bytes/sec
total size is 1130228  speedup is 0.94
```

Now view the nice new uploaded files:

```
duchess@pc:~$ rsync server1::backup_dir1/drawings/
drwx------    4,096  2021/01/04  06:06:55    .
-rw-r--r--   21,560  2021/09/17  08:53:18    aug_03
-rw-r--r--   21,560  2021/10/14  16:42:16    sept_03
```

Upload a few more files to the server, then download files to a different computer from the *rsyncd* server:

```
madmax@buntu:~$ rsync -av server1::backup_dir1/drawings ~/downloads
receiving incremental file list
created directory /home/madmax/downloads
drawings/
drawings/aug_03
drawings/sept_03

sent 123 bytes  received 11562479 bytes  1755.00 bytes/sec
total size is 1141776 speedup is 1.00
```

Everything works. Take a break and enjoy your success.

Discussion

This is not a secure form of file transfer because there is no encryption, and anyone on the network can access the files. It is suitable to use on your local network for easy archiving and file sharing.

rsync [hostname]:: needs double colons when connecting to an *rsync* server running in daemon mode. This tells *rsync* to look for a module name.

These are the command options in the */etc/rsyncd.conf* example:

[backup_dir1]
: The module name can be anything you want.

path =
: Defines the directory for the module to use.

comment =
: This is a brief description to remind you who the module belongs to, or what it is for.

list=yes
: Permits users to see a list of files in the module. *no* hides the module.

read only = no
: This allows users to upload files to the server.

use chroot = no
: Overrides the default of *use chroot = yes*. *chroot* is *change root*, sometimes called a *chroot jail*. A *chroot jail* is a separate environment inside your filesystem that contains its own root filesystem, commands, libraries, and everything else it needs to function. This is not a secure environment, though it is commonly thought of as a security tool. For *rsync*, the man page describes this as useful protection from configuration errors. The trade-off is *rsync* is blocked from following symlinks to

files outside of the chroot environment, and it complicates preserving UIDs and GIDs by name. As they say, your mileage may vary, and it may be a good option for you. See the *use chroot* section of *rsyncd.conf (5)*.

Set both *uid* and *gid* to *root* or *0*. This preserves UIDs and GIDs, and manages permissions correctly.

If any of your transfers fail, look at the *rsync* error messages. They will tell you if you made a mistake in your filepaths, misspelled something, or couldn't connect to the server, and give useful hints for correcting the problem.

If you are running a Linux without systemd, consult its documentation to learn how to start and stop rsyncd.

See Recipe 7.14 to learn how to set up access controls.

See Also

- *man 5 rsyncd.conf*

7.14 Limiting Access to rsyncd Modules

Problem

You don't want an open *rsyncd* server, and you want users to have their own protected modules that other users cannot access.

Solution

rsyncd comes with its own simple authentication and access controls. Create a new file containing username/password pairs, and add *auth users* and *secrets file* directives to */etc/rsyncd.conf*.

First create the password file. In the following example, */etc/rsyncd-users* sets up three users and their passwords:

```
# rsync-users for server1
duchess:12345
madmax:23456
stash:34567
```

Set permissions to read-write for root-only:

```
$ sudo chmod 0600 /etc/rsyncd-users
```

Now create a module for one of your users in */etc/rsyncd.conf*. This example creates a module for Duchess, using the */backups/duchess* directory on the *rsync* server:

```
[duchess_backup]
   path = /backups/duchess
   comment = Duchess's private archive
   list = yes
   read only = no
   auth users = duchess
   secrets file =/etc/rsyncd-users
   use chroot = no
   strict modes = yes
   uid = root
   gid = root
```

Create your user's backup directory, like this example for Duchess, with mode 0700:

```
$ sudo mkdir /backups/duchess/
$ sudo chmod -R 0700 /backups/duchess/
```

Now try logging in:

```
$ rsync duchess@server1::duchess_backup
Password: 12345
drwxr-xr-x      4,096 2020/06/29   18:24:43 .
```

Try transferring some files:

```
$ rsync -av ~/logs duchess@server1::duchess_backup
Password:
sending incremental file list
logs/
logs/irc.log
logs/irc_#core-standup.log
logs/irc_#core.log
logs/irc_#desktop.log
logs/irc_#engineering.log
logs/irc_#mobile.log

sent 130,507 bytes  received 305 bytes  37,374.86 bytes/sec
total size is 129,383  speedup is 0.99
```

It worked! If the file transfer fails, check the *rsync* log to learn why. On systemd Linuxes, read the most recent log entries in the status output:

```
$ systemctl status rsyncd.service
```

On other Linux distributions the rsyncd log should be in */var/log*.

Discussion

The username/password pairs are arbitrary and are not related to system user accounts. *rsyncd* users have no access to the host system outside of their *rsync* shares.

For additional security, add these directives to */etc/rsyncd.conf*:

hosts allow
: Use this to list hosts that are allowed to access the *rsyncd* archives. For example, you can limit access to hosts on a single subnet:

```
hosts allow = *.local.net
hosts allow = 192.168.1.
```

: All hosts not allowed are denied, so you don't need a *hosts deny* directive.

hosts deny
: This usually isn't needed, if you use *hosts allow*. It is useful for denying access to specific hosts that cause annoyance.

: The password file is in cleartext, so it must be restricted to the superuser.

See Also

- *man 5 rsyncd.conf*
- The Discussion in Recipe 7.13 to learn about the command options.

7.15 Creating a Message of the Day for rsyncd

Problem

You're running an *rsyncd* server, and you think it would be nice to greet users with a cheerful message.

Solution

Create your message of the day (MOTD) in a plain-text file, such as */etc/rsync-motd*:

```
Welcome to your local backup server! Please remember to actually back up
your files!
```

Then configure the MOTD file location at the top of */etc/rsyncd.conf*:

```
[global]
motd file = /etc/rsync-motd
```

When users connect to your server, they will see your message:

```
$ rsync server1::backup_dir1/
Welcome to your local backup server! Please remember to actually backup your
files!

drwx------          4,096 2020/06/29 18:24:43 .
-rwxr-xr-x          6,400 2015/03/13 08:21:21 keytool
drwx------          4,096 2020/06/17 06:07:41 WIP
drwx------          4,096 2020/06/17 06:06:55 bin
drwxr-xr-x          4,096 2020/06/30 09:47:42 duchess
[...]
```

Discussion

A message of the day is an old Unix tradition. Use it for cheery greetings, maintenance downtime announcements, security tips, backup tips, or anything you think is important.

See Also

- *man 5 rsyncd.conf*

CHAPTER 8

Managing Disk Partitioning with parted

All mass storage drives—SATA hard disks, solid state drives, USB drives, SD (Secure Digital), NVMe (Non-Volatile Memory Express), and CompactFlash cards—must be partitioned and formatted with filesystems before you can use them. They all ship with some kind of partitioning and filesystems, which may not be what you want. As your needs change, you will want to repartition your disks and use different filesystems. In this chapter you will learn about using *parted* (partition editor) to manage partitioning.

Overview

parted only manages partitioning; see Chapter 11 to learn about filesystems. Chapter 9 covers the graphical frontend to *parted*, GParted, which manages both partitioning and filesystems.

You will also learn about the modern replacement for the Master Boot Record (MBR), which is the elderly and inadequate legacy partition table. The MBR has been supplanted by the new Globally Unique Identifier Partition Table (GUID Partition Table or GPT).

parted shows partition information and adds, removes, and resizes partitions. *parted* has just one gotcha: it writes your changes to disk immediately, so you must be careful. GParted does not apply changes until you click a button.

It is a common convenience to call all mass storage devices *disks*, even though many of them are not disks anymore, but solid-state devices, like USB sticks. Why not, when we still dial telephones, and make tape recordings and film videos with our smartphones?

A disk partition is a logical division of a storage disk, a way of dividing the disk into one or more independent regions. A disk must have at least one partition. The number of partitions depends on your needs and whims. After partitioning a disk, you must put a filesystem on each partition, and then you can use it. A single disk may have multiple partitions, and each partition may have a different filesystem.

The disk name on Linux is always */dev* something, which is short for device. For example, */dev/sda* for a hard disk, and */dev/sr0* for an optical drive. Partitions are the disk name plus a number. If */dev/sda* has three partitions, they are */dev/sda1*, */dev/sda2*, and */dev/sda3*.

Partitioning Schemes

The default partitioning scheme on some Linux distributions is to stuff the whole installation into a single partition. This works fine, but setting up a few more partitions during installation has some advantages:

- Giving */boot* its own partition makes managing multiboot systems easier because the boot files are independent of whatever operating systems you install or remove.
- Put */home* on its own partition to isolate it from the root filesystem, so you can replace your Linux installation without touching */home*. */home* could even be on a separate drive.
- */var* and */tmp* can fill up from runaway processes. Putting them on their own partitions prevents them from interfering with the other filesystems.
- Putting the swap file on its own partition enables suspend-to-disk.

See Chapter 1 to learn more about designing your partitioning layouts.

Partition Tables: GPT and MBR

The GUID Partition Table (GPT), first released in 2010, is the modern replacement for the antique PC-DOS Master Boot Record (MBR). If your only experience is with the MBR, prepare yourself for a treat, because the GPT is a big improvement.

The MBR was created for IBM PCs way back in the last millennium in the early 1980s, during the exciting era of 10-megabyte (MB) hard disks. The MBR goes on the first 512 bytes of the first sector of your disk, preceding the first partition, and holds the bootloader and partition table. The bootloader occupies 446 bytes, the partition table uses 64 bytes, and the remaining 2 bytes store the boot signature.

64 bytes is not much room to store much of anything, so the MBR is limited to four primary partitions. One primary partition may hold an extended partition, which can then be divided into logical partitions. Linux supports (theoretically) an unlimited

number of logical partitions. Even with great thundering herds of logical partitions, the MBR is limited to addressing a maximum disk size of 2.2 TiB, which these days is barely enough to hold your cat memes. Why this limitation? You can do the math yourself: the MBR is limited to 32 bits of addressing, and can address 2^{32} number of blocks (we'll discuss blocks and sectors in a moment), so the equation for disks with 512-byte blocks is 2^{32} x 512 = $2.199023256 \times 10^{12}$ bytes.

BIOS and UEFI

The GPT is part of the UEFI (Unified Extensible Firmware Interface) specification. UEFI replaces your computer's Basic Input Output System, better known as the PC BIOS, or just plain BIOS. Figure 8-1 is the old legacy BIOS, and Figure 8-2 is a modern UEFI, all full of shiny whizbang features, just like a little operating system.

GPT has many advantages over the MBR:

- Up to 128 partitions on Linux, numbered 1–128, and no messing with primary and extended partitions
- Fault-tolerance: copies of the partition table are stored in multiple locations
- Unique IDs for disks and partitions
- Legacy BIOS/MBR boot mode
- Verifies its own integrity and the partition table
- Secure Boot

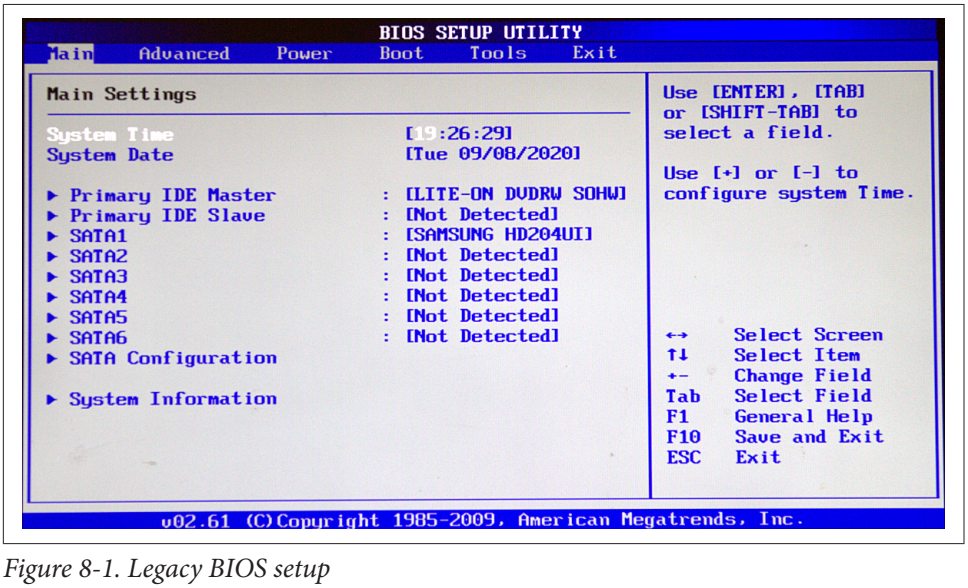


Figure 8-1. Legacy BIOS setup

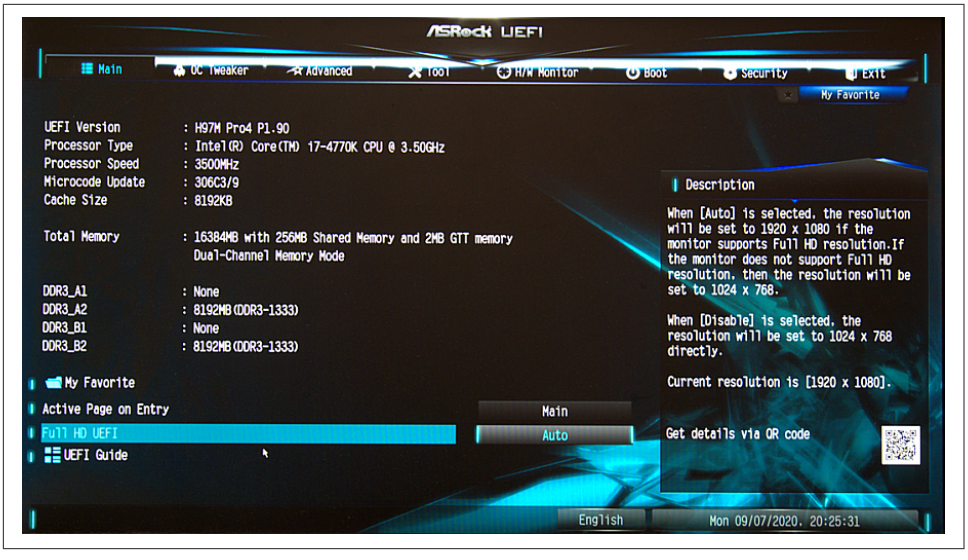


Figure 8-2. UEFI setup

The MBR is nearly obsolete, and you should use the GPT. In GPT, the first sector of the disk is reserved for a protective MBR that supports GPT on a BIOS computer, so we can use the GPT on older systems that have a BIOS instead of UEFI. The bootloader and operating system must both be GPT-aware, which has been the case with Linux for years. The only reason to use the MBR is on old computers with old operating systems that do not support the GPT.

If you have an older system with a BIOS, you cannot upgrade it to UEFI, but must replace the motherboard to get UEFI. Both UEFI and BIOS are integrated into the motherboard.

Blocks and Sectors

Now we will talk about blocks and sectors, and how they affect the maximum sizes of your disks, files, and partitions. *Blocks* are the smallest storage units on a disk that a filesystem can use. These are logical, not physical, divisions. The smallest physical unit of storage is a *sector*. Blocks can span multiple sectors, and a file can span multiple blocks.

When a file spans multiple blocks, there is a certain amount of waste because files rarely match block sizes. For example, a file that is one byte larger than four blocks uses five blocks. The fifth block holds just that one byte, and that block is exclusive to the file. Because of this you might think that 512-byte blocks are less wasteful. But there is more information stored in a block than just the file.

Every block, in addition to your file data, stores timestamps, the filename, ownership, permissions, the block ID, and its correct order with other blocks, the inode, and other metadata.

4096-byte blocks use one-eighth the metadata of 512-byte blocks. On a 4 TiB hard disk, you need 8,000,000,000 512-byte blocks. With a 4096-byte block size there are only 1,000,000,000 blocks, which represents quite a lot of metadata savings.

The sector size limits the size of storage volumes. The standard sector size for hard disks has been 512 bytes for some years, and now 4096 bytes is the standard because hard disks have grown so large.

The GPT provides 64-bit addressing, supporting 2^{64} total blocks on a single disk, so a hard disk with 512-byte blocks can be as large as 9 zettabytes. With 4096-byte blocks, your maximum disk size is 64 zettabytes, which I daresay is sufficient for even the most dedicated cat meme collector. These are theoretical maximums, limited by available hardware, operating system limits, and filesystem support for large volumes. For example, the Ext4 filesystem maxes out at 1 EiB for a single filesystem, and a maximum 16 TiB file size with a 4096-byte block size. XFS supports a maximum filesystem and file size of 8 EiB minus 1 byte.

CDs and DVDs have 2048-byte sectors. Solid-state devices such as USB sticks, SD cards, CompactFlash, and Solid State Drives (SSDs) also have sectors and blocks. The smallest unit on an SSD is called a *page*. Common page sizes are 2 KB, 4 KB, 8 KB, and larger. Blocks contain 128 to 256 pages, and block size is typically 256 KB to 4 MB.

All of these enormous numbers are a bit dizzying. Table 8-1 summarizes the decimal and binary measurements used to measure disk capacity.

Table 8-1. Decimal and binary multiples of bytes

Value	Decimal	Value	Binary
1	B byte	1	B byte
1000	kB kilobyte	1024	KiB kibibyte
1000^2	MB megabyte	1024^2	MiB mebibyte
1000^3	GB gigabyte	1024^3	GiB gibibyte
1000^4	TB terabyte	1024^4	TiB tebibyte
1000^5	PB petabyte	1024^5	PiB pebibyte
1000^6	EB exabyte	1024^6	EiB exbibyte
1000^7	ZB zettabyte	1024^7	ZiB zebibyte
1000^8	YB yottabyte	1024^8	YiB yobibyte

The decimal values are powers of 10; for example, a kilobyte is 1000 bytes, or 10^3. The binary values are powers of two, so a kibibyte is 2^{10}, 1024 bytes. Hard disk manufacturers like to use the decimal format to make their drives look bigger.

Whoever came up with the weird "bibyte" naming scheme just about guaranteed that nobody would ever want to say the names. It's all a mishmash anyway, as people like to use them interchangeably. At any rate, now you know the difference.

8.1 Unmounting Your Partitions Before Using parted

Problem

You know you must unmount your partition, or partitions, before you can make any changes with *parted*, and you need to know how.

Solution

Unmount a partition from your graphical file manager, or use the *umount* command. The following example unmounts */dev/sdc2*:

```
$ sudo umount /dev/sdc2
```

How do you know the correct device name? See Recipe 8.3 to learn how to list your attached disks and partitions.

If you are creating a new partition table on a disk, you should unmount all the partitions on it.



Changing a Running System

It is risky to unmount filesystems attached to the active root filesystem, such as */home*, */var*, or */tmp*, if they are on separate partitions. It is safer to perform partitioning operations from another Linux instance, such as SystemRescue (Chapter 19), or a second Linux on the same machine (Chapter 1).

Discussion

Technically, you mount and unmount filesystems rather than partitions. However, I shall not hold it against you if you say "partitions."

See Also

- *man 8 parted*
- Parted User's Manual

8.2 Choosing the Command Mode for parted

Problem

You know you can start the *parted* command in interactive mode, launching the *parted* command shell, or run it as an ordinary command, and you want to know how to do both.

Solution

Running *parted* with no options launches the interactive *parted* shell. You need root privileges:

```
$ sudo parted
GNU Parted 3.2
Using /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted)
```

When your normal command prompt changes to *(parted)*, you are in the *parted* shell. Type **`help`** to see a list of commands and their descriptions. There is also help for the individual *parted* commands, for example, *help print*. Type *quit* to exit parted. Most *parted* commands can be abbreviated to their first letter, like *h* and *q*.

Enter a complete command to run *parted* as a normal command in your regular shell, like this example that lists all of your disks:

```
$ sudo parted /dev/sdb print devices
/dev/sdb (2000GB)
/dev/sda (4001GB)
/dev/sdc (4010MB)
/dev/sdd (15.7GB)
/dev/sr0 (425MB)
```

The command runs and exits, and returns to your normal command prompt.

Discussion

Be careful in both modes, because *parted* applies your changes immediately. Always have good backups before doing anything with *parted*.

See Also

- *man 8 parted*
- Parted User's Manual

8.3 Viewing Your Existing Disks and Partitions

Problem

You want to see your existing partitions, their sizes, and what filesystems are on them.

Solution

If you don't know the names of the disks on your system, run *parted* with no options:

```
$ sudo parted
GNU Parted 3.2
Using /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted)
```

When you have not selected a device, *parted* guesses which one you want, usually the first one, and tells you which one it has selected (see *Using /dev/sda* in the preceding example).

print devices lists your disk names and sizes:

```
(parted) print devices
/dev/sda (256GB)
/dev/sdb (1000GB)
/dev/sdc (4010MB)
```

Select which device you want to look at, then display its information:

```
(parted) select /dev/sdb
Using /dev/sdb
(parted) print
Model: ATA ST1000DM003-1SB1 (scsi)
Disk /dev/sdb: 1000GB
Sector size (logical/physical): 512B/4096B
Partition Table: gpt
Disk Flags:

Number  Start   End     Size    File system     Name  Flags
 1      1049kB  525MB   524MB   fat16                 boot, esp
 2      525MB   344GB   343GB   btrfs
 3      344GB   998GB   654GB   xfs
 4      998GB   1000GB  2148MB  linux-swap(v1)        swap

(parted)
```

Type **`quit`** to exit.

You can open the *parted* shell to a specific disk:

```
$ sudo parted /dev/sda
GNU Parted 3.2
```

```
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
```

Enter **`print`** with no options to see information about this disk:

```
(parted) print
 Model: ATA SAMSUNG SSD SM87 (scsi)
Disk /dev/sda: 256GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
[...]
```

print all lists all partitions on all devices:

```
(parted) print all
 Model: ATA SAMSUNG SSD SM87 (scsi)
Disk /dev/sda: 256GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start   End    Size    File system  Name                  Flags
 1      1049kB  524MB  523MB   fat16        EFI system            legacy_boot,
                                            partition             msftdata

 2      524MB   659MB  134MB                Microsoft reserved    msftres
                                            partition
 3      659MB   253GB  253GB   ntfs         Basic data partition  msftdata

 4      253GB   256GB  2561MB  ntfs                               diag

Model: ATA ST1000DM003-1SB1 (scsi)
Disk /dev/sdb: 1000GB
Sector size (logical/physical): 512B/4096B
Partition Table: gpt
Disk Flags:

Number  Start   End     Size    File system     Name  Flags
 1      1049kB  525MB   524MB   fat16                 boot, esp
 2      525MB   344GB   343GB   btrfs
 3      344GB   998GB   654GB   xfs
 4      998GB   1000GB  2148MB  linux-swap(v1)        swap

Model: General USB Flash Disk (scsi)
Disk /dev/sdc: 4010MB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
 1      1049kB  4010MB  4009MB  primary  fat32
```

Find any unpartitioned free space on any disk:

```
(parted) print free
Model: ATA ST4000DM000-1F21 (scsi)
Disk /dev/sda: 4001GB
Sector size (logical/physical): 512B/4096B
Partition Table: gpt
Disk Flags:

Number  Start   End     Size    File system     Name  Flags
        17.4kB  1049kB  1031kB  Free Space
 1      1049kB  500MB   499MB   ext4
 2      500MB   60.5GB  60.0GB  ext4
 3      60.5GB  2061GB  2000GB  xfs
 4      2061GB  2069GB  8000MB  linux-swap(v1)
        2069GB  4001GB  1932GB  Free Space
```

Discussion

Let's take a look at what all of this output means:

- *Model* is the manufacturer's name for the device.
- *Disk* gives the device name and size.
- *Sector size* gives both the logical and physical block size. A logical block size of 512B is for backward compatibility with older disk controllers and software.
- *Partition table* tells you the partition type, either *msdos* or *gpt*.
- *Flags* matter more to Windows than Linux. They identify the partition types, and in some cases are necessary so Windows gets less confused. The full list is in the *Parted User's Manual*.

These are the partition flags in the examples:

- *legacy_boot* marks a GPT partition as bootable.
- *msftdata* labels GPT partitions that contain Microsoft filesystems, either NTFS or FAT.
- *msftres* is a Microsoft reserved partition. This is a special partition that is required by Microsoft on GPT partitions, for use by the operating system. On partitions less than 16 GB in size, the MSR is 32 MB, and on larger drives it is 128 MB.
- *diag* is a Windows recovery partition.
- *boot, esp* both mark the partition as a boot partition. *boot* is an MBR label, and *esp* is a GPT label.
- *swap* marks swap partitions.

See Also

- *Parted User's Manual*
- *man 8 parted*

8.4 Creating GPT Partitions on a Nonbooting Disk

Problem

You want to repartition a disk, removing all data and starting over with a new GUID Partition Table (GPT). This is not a bootable disk with an operating system, but is only for data storage.

Solution

First, create the new partition table, then create your partitions, then verify that all were created correctly. Be very certain that you select the correct disk; see Recipe 8.3 to learn how to list your disks and partitions.

In the following example, there is a USB stick at */dev/sdc*, which is used for data storage. It is not a bootable disk with an operating system on it. You must unmount your devices before running *parted*. The first step is to unmount it, then create a new GPT partition table:

```
$ sudo umount /dev/sdc
$ sudo parted /dev/sdc
GNU Parted 3.2
Using /dev/sdc
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) mklabel gpt
Warning: The existing disk label on /dev/sdc will be destroyed and all data on
this disk will be lost. Do you want to continue?
Yes/No? Yes
(parted) p
Model: General USB Flash Disk (scsi)
Disk /dev/sdc: 4010MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start  End  Size  File system  Name  Flags
```

Now you can create new partitions. The following example creates two partitions that are about the same size. You must specify a name for the partition, and the start and end locations for both partitions:

```
(parted) mkpart "images" ext4 1MB 2004MB
(parted) mkpart "audio files" xfs 2005MB 100%
```

Then check your work, and exit:

```
(parted) print
Model: General USB Flash Disk (scsi)
Disk /dev/sdc: 4010MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start   End     Size    File system  Name         Flags
 1      1049kB  2005MB  2004MB  ext4         images
 2      2006MB  4009MB  2003MB  xfs          audio files

(parted) q
Information: You may need to update /etc/fstab.
```

If your start or end points are too close to another partition, you will see an error message. In the following example, the start of the second partition is the same as the end of the first partition:

```
(parted) mkpart "images" ext4 2004MB 100%
Warning: You requested a partition from 2004MB to 4010MB (sectors
3914062..7831551).
The closest location we can manage is 2005MB to 4010MB (sectors
3915776..7831518).
Is this still acceptable to you?
Yes/No? Yes
```

Changing it to 200 5MB fixes the error.

Discussion

start sets the beginning of the new partition. This is always a number value. The `1MB` value in the example means one megabyte from the beginning of the disk. You cannot start at zero because the first 33 sectors are reserved for the EFI label, so the first partition starts at the 34th sector or higher. I start at the one megabyte mark because it is easy to remember.

end can take a size value or a percentage. In the example, the end of first partition is 200 5MB from the start of the first partition. The second partition ends at 100% of the remaining space. Creating a new partition table wipes out all data on the disk.

You must put filesystems on your new partitions before they are usable (see Chapter 11).

The warning "You may need to update /etc/fstab" applies only if you change partitions that are in your */etc/fstab* file.

The syntax for creating new GPT partitions is *mkpart name fs-type start end*.

name is required. This is anything you want, so you can make it a name that helps you remember what the partition is for.

The *fs-type* label is not required, but you should specify it so that the partition is assigned the correct filesystem type code. Run *help mkpart* in the *parted* shell to see a list of filesystem labels.

Even though you created filesystem labels, there are no filesystems on your disk. Creating filesystems is a separate step.

The filesystem labels sometimes disappear. After you put a filesystem on the partition, they will stay put.

The *parted* help and documentation are a bit confusing on the differences between creating GPT partitions and MS-DOS partitions. When you create a GPT partition, you must create a *name* for it. When you create an MS-DOS partition you must specify a *part-type*, which is one of *primary*, *extended*, or *logical*. There is a fair bit of confusion about this, and the result is admins creating GPT partition names of *primary*, *extended*, and *logical*. This is not correct and you should create *names* for GPT partitions.

At any rate, you should not create MS-DOS partition tables because they are obsolete, except on old computers with old software that does not support GPT.

See Also

- *Parted User's Manual*
- *man 8 parted*
- Chapter 11

8.5 Creating Partitions for Installing Linux

Problem

You want to install Linux on a disk and need to know how to partition it.

Solution

Use the partition manager in the Linux installer. You can set up your partitions before running the installer, but using the installer's partition manager ensures that it will be done correctly, and you will see warnings for any errors. See Recipe 1.8 for a suggested partitioning scheme.

Discussion

Most Linux installers provide guidance for partitioning for a new installation and also allow manual customizations.

See Also

- The introduction to this chapter for partitioning suggestions
- Chapter 1

8.6 Removing Partitions

Problem

You want to delete some partitions.

Solution

Start *parted* in interactive mode for the disk you want to make changes on, then print the partition table:

```
$ sudo parted /dev/sdc
GNU Parted 3.2
Using /dev/sdc
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) p

Model: General USB Flash Disk (scsi)
Disk /dev/sdc: 4010MB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
 1      1049kB  2005MB  2004MB  primary
 2      2005MB  4010MB  2005MB  primary
```

In this example, delete the second partition by typing *rm 2*. The partition will be immediately removed, and there will not be a confirmation. Then type *p* to verify:

```
(parted) rm 2
(parted) p
Model: General USB Flash Disk (scsi)
Disk /dev/sdc: 4010MB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:
```

```
Number  Start   End     Size    Type     File system  Flags
 1      1049kB  2005MB  2004MB  primary
```

Discussion

Be very certain you are deleting the correct partitions. It is OK to make written notes and check many times before you start.

If you try to delete a mounted partition, *parted* will warn you with "Warning: Partition /dev/sdc2 is being used. Are you sure you want to continue?" You may go ahead and delete it. Any open files will remain in memory until you reboot or close them, which is kind of fun because you can still read and save the files to a different partition.

See Also

- *man 8 parted*
- Parted User's Manual

8.7 Recovering a Deleted Partition

Problem

You deleted a partition, and now you wish you hadn't, and you want to get it back.

Solution

If you accidentally deleted a new empty partition, don't bother trying to recover it, just create it again. If your partition had a filesystem and data on it, then your best chance is to try immediate recovery. In the *parted* shell, use the *rescue* command, and give it the partition's start and end locations. These can be approximate:

```
(parted) rescue 2000MB 4010MB
searching for file systems... 40%       (time left 00:01)Information: A ext4
primary partition was found at 2005MB -> 4010MB.  Do you want to add it to the
partition table?
Yes/No/Cancel? Yes
```

parted won't give you any feedback, so print the partition table to see if the lost partition came back:

```
(parted) p
Model: General USB Flash Disk (scsi)
Disk /dev/sdc: 4010MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:
```

```
Number  Start   End     Size    File system  Name    Flags
 1      1049kB  2005MB  2004MB  xfs          images
 2      2005MB  4010MB  2005MB  ext4
```

And there it is. With a little luck all of your files are intact.

Discussion

The longer you wait to try to restore a partition, the more likely it will not be restorable because it may be unintentionally overwritten. If you need to delay rescue operations until a later time, put it away in a safe place, if possible.

As always, your best practice is to always maintain good backups.

See Also

- *Parted User's Manual*
- *man 8 parted*

8.8 Increasing Partition Size

Problem

You want to increase the size of an existing partition, which has a filesystem on it.

Solution

The following example increases the size of a partition with a filesystem on it. There are two steps: first resize the partition, then resize the filesystem to match. Every filesystem has its own set of tools, and you must use the correct tool for increasing the size. In this recipe, we will resize the Ext4, XFS, Btrfs, and FAT16/32 partitions.

Ext4, XFS, and Btrfs can all be enlarged online or offline. FAT16/32 can be resized only offline and must be unmounted first.

There must be free space at the end of the partition you want to increase. Open the *parted* shell to your selected disk, and look for free space:

```
$ sudo parted /dev/sdc
GNU Parted 3.2
Using /dev/sdc
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) print free
Model: General USB Flash Disk (scsi)
Disk /dev/sdc: 4010MB
Sector size (logical/physical): 512B/512B
```

```
Partition Table: gpt
Disk Flags:

Number  Start   End     Size    File system  Name    Flags
[...]
        1024MB  2005MB  981MB   Free Space
 2      2005MB  3500MB  1495MB  ext4         audio
        3500MB  4010MB  510MB   Free Space
```

This shows 981 MB of free space preceding Partition 2, and 510 MB of free space following. You can only change the end point of a partition, so the following examples expand Partition 2 to use all of the 510 MB of free space at the end.

First, expand the partition to its new end point:

```
(parted) resizepart 2 4010MB
```

You will not see a success message, but if you make a mistake, you will see an error message. Type **p** to see the partition table and verify that *resizepart* did what you want.

Now you must expand the filesystem to fit the new partition size with the appropriate command for the filesystem. Table 8-2 shows the commands to use for each filesystem, expanding them to fill their partitions.

Table 8-2. Commands to increase filesystem sizes

Filesystem	Resize command
Ext4	sudo resize2fs /dev/sd*c2*
XFS	sudo xfs_growfs -d /dev/sd*c2*
Btrfs	sudo btrfs filesystem resize max /dev/sd*c2*
FAT16/32	sudo fatresize -i /dev/sd*c2*

Remember that FAT16/32 must be unmounted first.

Print the partition table in *parted* to check your work.

Discussion

The examples in this chapter and in Recipe 8.9 are small, using a 4 GB USB stick. This is great for testing, but in real life you will likely be using larger disks. The commands are the same, except for partition sizes.

As always, you should have current backups before you start.

You could resize a filesystem to be smaller than the partition, but that doesn't make sense. Check out Chapter 11 to learn all about creating and managing filesystems.

If you are wondering "Where is my favorite filesystem?" I chose Ext4, Btrfs, XFS, and FAT16/32 because those are the most commonly used Linux filesystems, and they are all well maintained.

See Also

- Chapter 11
- *man 8 resize2fs*
- *man 8 parted*
- *man 8 xfs_growfs*
- *man 8 btrfs*
- *man 8 fsck.vfat*

8.9 Shrinking a Partition

Problem

You have a partition with a filesystem on it, and you want to shrink it.

Solution

XFS filesystems cannot be reduced in size, only increased. You can shrink Ext4, Btrfs, and FAT16/32. Ext4 and FAT16/32 must be unmounted before shrinking them. Btrfs can be shrunken online, but it is safer to unmount it first.

Make sure that the used portion of the filesystem you want to shrink is smaller than the size you want to shrink it to. Use the *du* command to see how much space your files occupy:

```
$ du -sh /media/duchess/shrinkme
922.6M    /media/duchess/shrinkme
```

You should allow about 40% extra room for metadata, wasted block space, and for just in case, so in this example the new size should not be smaller than 1.4 GB. If you need room to add more files, then account for that as well.

Shrinking partitions is a little more complicated than expanding them. There are more steps, and the filesystems must be shrunk offline. If the partition is on an external storage device, such as a USB stick, unmount it and then shrink it. If it is a partition that belongs to your running system, then you must run *parted* from a bootable rescue disk, or a second Linux on a multiboot system, so that you can unmount the filesystem you want to shrink.

After your selected filesystem is unmounted, follow these steps:

- Run a filesystem check
- Shrink the filesystem
- Shrink the partition

Run the following command to check the health of an Ext4 filesystem:

```
$ sudo e2fsck -f /dev/sdc2
```

Check a Btrfs filesystem:

```
$ sudo btrfs check /dev/sdc2
```

Check a FAT16/32 filesystem:

```
$ sudo fsck.vfat -v /dev/sdc2
```

When everything checks out, shrink your filesystem. The examples in Table 8-3 shrink the filesystems to 2000 MB.

Table 8-3. Commands to decrease filesystem sizes

Filesystem	Resize command
Ext4	sudo resize2fs /dev/sd*c2* 2g
Btrfs	sudo btrfs filesystem resize 2g /dev/sd*c2*
FAT16/32	sudo fatresize -s 2G /dev/sd*c2*

Now you can shrink your partition to match the filesystem size. Open the *parted* shell to your device and then run the *resize* command. Specify the partition number and the end point:

```
(parted) resizepart 1 2000MB
Warning: Shrinking a partition can cause data loss, are you sure you want to
continue?
Yes/No? y
```

Check your work by printing the partition table in *parted*.

Discussion

Storage media is large and cheap. In the olden days, fiddling with partitions was necessary for cramming the most files onto a disk. Now we have the luxury of customizing their sizes for our convenience.

See Also

- Chapter 11
- *man 8 resize2fs*
- *man 8 parted*
- *man 8 btrfs*
- *man 8 fsck.vfat*

CHAPTER 9

Managing Partitions and Filesystems with GParted

GParted, the GNOME Partition Manager, is one of my favorite tools on Linux. GParted is a nice graphical front-end to the *parted* partition manager command and all of the filesystem management commands. You can create, delete, move, copy, and resize partitions and filesystems, and create new partition tables with just a few clicks. Other features are data rescue and managing labels and UUIDs.

Labels on partitions and filesystems are useful for identifying partitions and filesystems in a friendly way, and to give filesystems short, easy names. Without a label, a filesystem is identified by its long UUID. For example, when you plug in a USB stick without filesystem labels, it appears as something like */media/username/1d742b2d-a621-4454-b4d3-469216a6f01e*. Give it a nice short label like *mystuff*, and then it mounts as */media/username/mystuff*.

After an operation in GParted is complete, the status window offers you the option to save a logfile of what it did. Save this information and study it because it shows you the commands it used.



Changing Your Running System

For some operations, such as copying, checking and repair, and setting labels and UUIDs, it is required to unmount the filesystems first. You cannot unmount the filesystems that are required for your running system. In this case, use a bootable SystemRescue CD/USB (Chapter 19). If you are running a multiboot system with more than one Linux distribution installed, boot to a different Linux and run GParted from there (see Chapter 1).

GParted requires root privileges. When you launch GParted, a dialog for entering your sudo or root password will open (Figure 9-1).

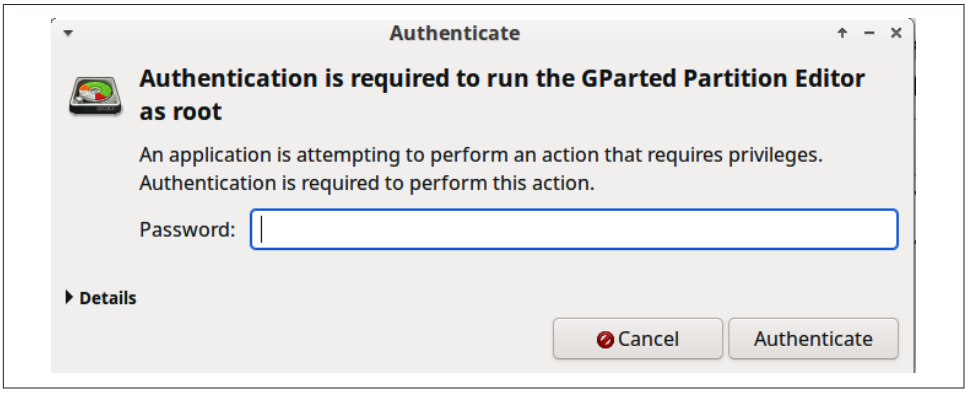


Figure 9-1. The application launcher asks for a password

It is common to call all storage media *disks*, even solid-state media like SSDs, USB drives, SD (Secure Digital), NVMe (Non-Volatile Memory Express), and CompactFlash. GParted manages any of these disks physically attached to your system, internal and external.

If you are not familiar with the basics of partitioning and managing filesystems, the introduction to Chapter 8 provides a detailed overview.



Be Careful!

Before you try any of the recipes in this chapter, have current backups and be very certain that you are operating on the correct disks and partitions.

Creating a new partition table wipes out your entire disk.

Deleting or otherwise damaging a partition loses all the data on that partition. It is possible to recover it, but not guaranteed.

USB sticks are great for practice and testing.

9.1 Viewing Partitions, Filesystems, and Free Space

Problem

You want to see all the partitions, filesystems, and free space on all attached disks.

Solution

Launch GParted, and use the drop-down menu on the upper right to see all attached disks (Figure 9-2). Click View → Device Information to open the panel on the left to see disk information, such as the model, serial number, size, and partition table type.

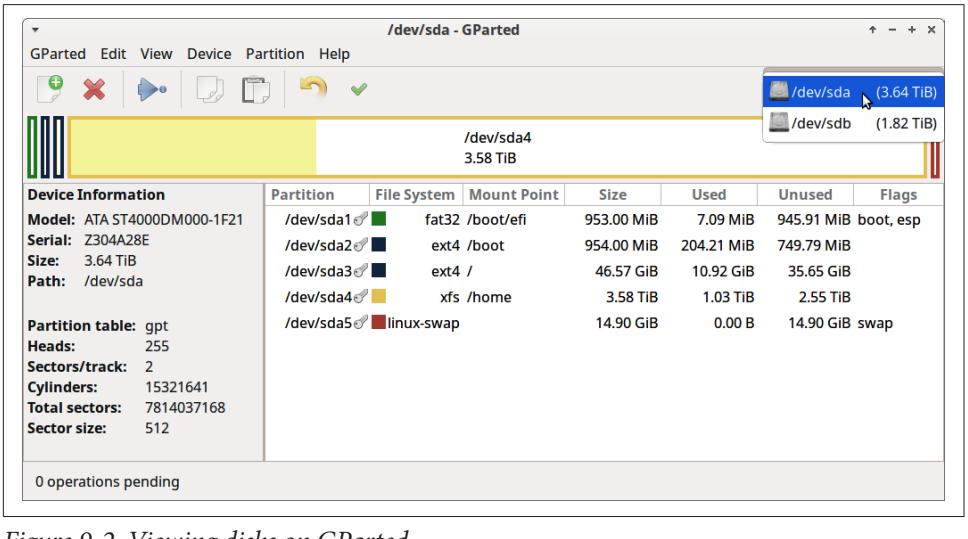


Figure 9-2. Viewing disks on GParted

You will see a lot of information: device names, mountpoints, filesystems, labels, partition type and sizes, used and total space, and free space. Right-click on any partition to open the operations menu, and click the Information button at the bottom of the menu to see more information on that partition (Figure 9-3).

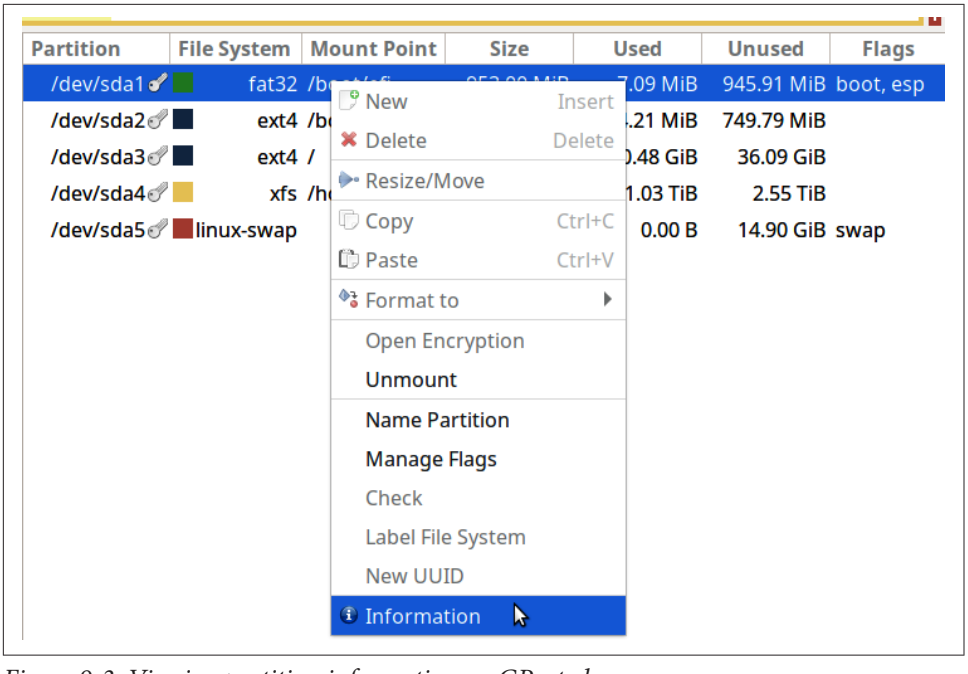


Figure 9-3. Viewing partition information on GParted

Discussion

GParted does not apply changes until you click the green checkmark in the top toolbar, so it is safe to poke around and explore. If you accidentally click a command, click the little curvy yellow arrow next to the checkmark to undo.

When you open the right-click operations menu, some commands are grayed out, because they can be used only on unmounted filesystems. Click the Unmount command, and then these commands will become available. Note that any filesystems that are necessary for your running system cannot be unmounted; in this case, use a SystemRescue CD/USB (Chapter 19).

See Also

- GNOME Partition Editor

9.2 Creating a New Partition Table

Problem

You want to reformat a disk with a new GPT partition table. Your existing partition table is MS-DOS, and you want to replace it with GPT, or it is a used disk with old installations on it, and you want to start over with a clean disk.

Solution

First, be very sure which disk you want to create the new partition table on, because this will erase all data on the disk. This is one operation that GParted applies immediately, after just one warning, and there is no Undo, so be careful.

Select your disk in the top-right dropdown menu, then click Device → Create Partition Table (Figure 9-4).

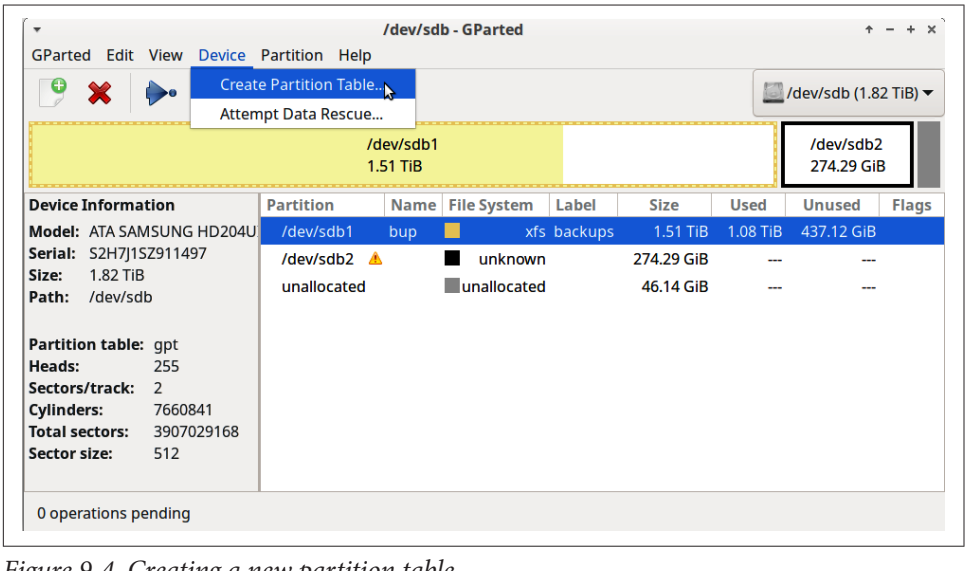


Figure 9-4. Creating a new partition table

Select the GPT partition table type and click Apply (Figure 9-5) .

It doesn't take very long, and then you will have a new, clean, empty disk, all ready to partition and format with new filesystems.

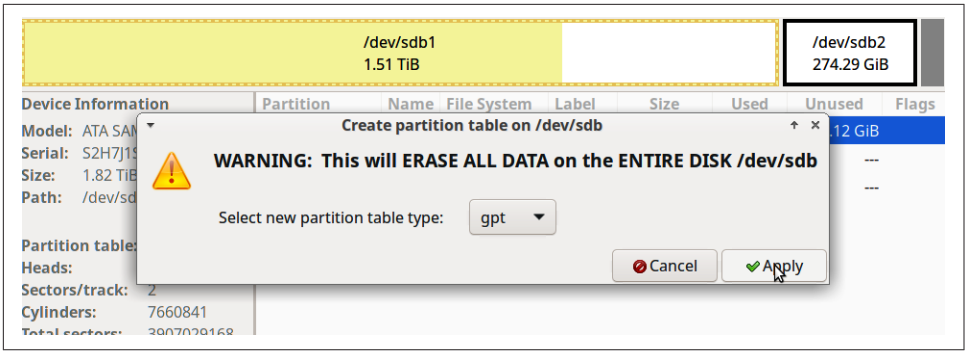


Figure 9-5. Select the partition table type

Discussion

Always create a GPT partition table, unless you have a reason to use something else. GParted supports several partition table types, including MS-DOS, BSD, Amiga, and AIX. GPT and MS-DOS are the most commonly used on the x86 platform. GPT is for modern large hard disks and is easier to manage and more resilient than the old MS-DOS partition table. See the introduction to Chapter 8 for detailed information on partition tables.

See Also

- GNOME Partition Editor
- Chapter 8

9.3 Deleting a Partition

Problem

You need to delete one or more partitions.

Solution

Select the partition you want to delete, and right-click to open the operations menu. If it has a mounted filesystem on it, you must first unmount it, which you can do by clicking Unmount in the menu. Then click Delete, click the green checkmark, and it is gone (Figure 9-6).

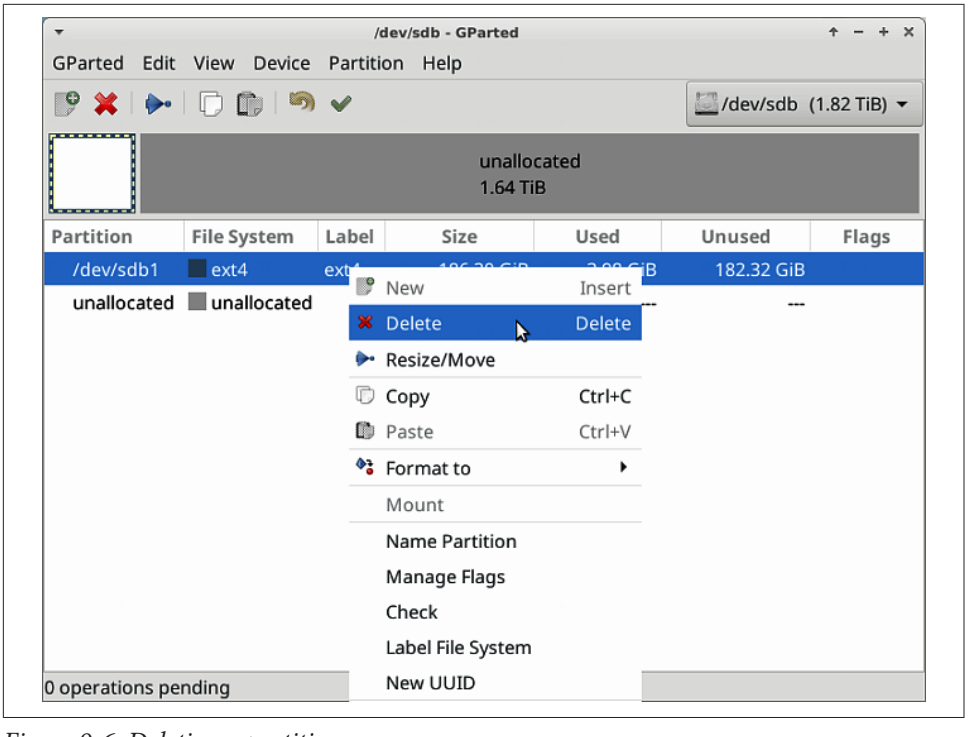


Figure 9-6. Deleting a partition

You will see a status message when the delete has finished.

Discussion

Deleting the partition deletes everything inside the partition, so if you have a filesystem and data on the partition, be very sure you want to delete it.

See Also

- GNOME Partition Editor
- Recipe 8.6

9.4 Creating a New Partition

Problem

You want to create new partitions.

Solution

All you need is empty space on a disk. The following example creates a new 400 GB partition and formats it with an Ext4 filesystem (Figure 9-7).

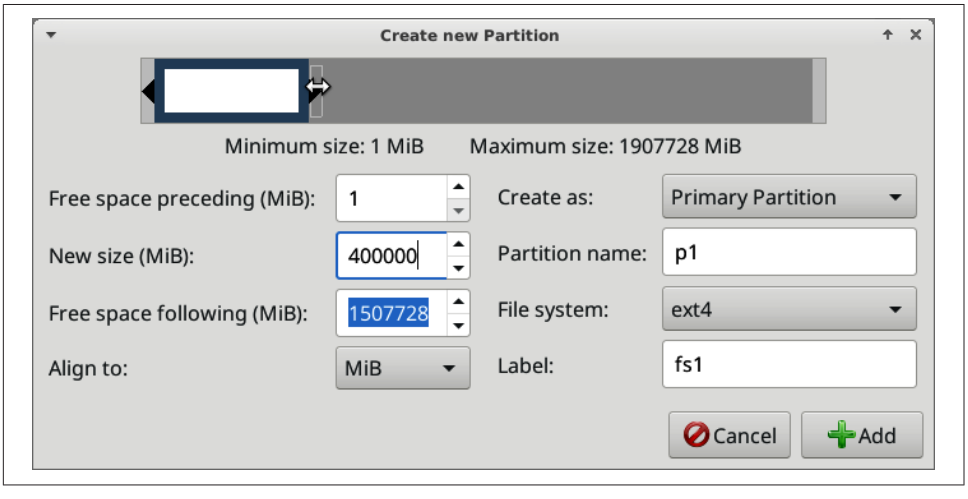


Figure 9-7. Creating a new partition

Click Partition → New in the top menu. This opens a new window, where you enter your partition size, select your filesystem, and create partition and filesystem labels. Use either the slider or the New Size (MiB) field to set your filesystem size. The values in the New Size field are in mibibytes, so 400,000 is 400 GiB. Then click Add, then the green checkmark.

When finished, see Chapter 6 to learn how to set correct ownership and permissions on your new filesystem.

Discussion

With a GPT partition table, you will always create only primary filesystems. The other two options, logical partition and extended partition, are only for MS-DOS partition tables. If you're not sure which one your disk is using, click View → Device Information. This opens a pane on the left with information about your disk, including the partition table type.

In the filesystem selector you also have the option to create an empty partition without a filesystem. This is way down at the bottom, Unformatted. Next to Unformatted is Clear, which deletes an existing filesystem and preserves the partition.

GParted combines creating a partition and putting a filesystem on it into a single, fast operation. This is faster than using *parted*, which only creates partitions and rquires you to create your filesystem separately.

See Also

- GNOME Partition Editor
- Chapter 8
- Chapter 11

9.5 Deleting a Filesystem Without Deleting the Partition

Problem

You want to remove a filesystem without deleting its underlying partition because you want to format the partition with a different filesystem, or the existing filesystem is corrupt and you need to reformat it and then copy your files back into it (Figure 9-8).

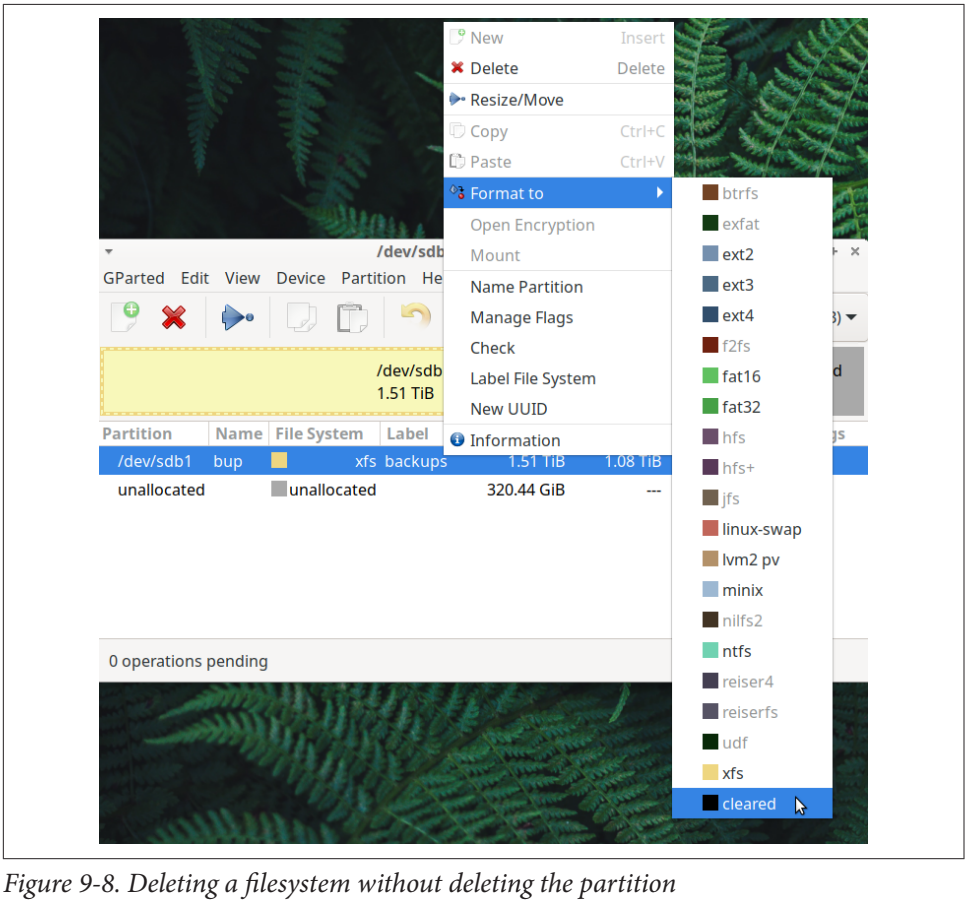


Figure 9-8. Deleting a filesystem without deleting the partition

Solution

The filesystem must first be unmounted. Right-click on the partition to open the operations menu, then click Unmount. When that is completed, click Format To. Scroll to the bottom of the list and click Cleared. This deletes the filesystem without deleting the partition.

See Also

- GNOME Partition Editor

9.6 Recovering a Deleted Partition

Problem

You deleted a partition, and now you wish you hadn't, and you want to get it back.

Solution

If you accidentally deleted a new empty partition, don't bother trying to recover it, just create it again. If your partition had a filesystem and data on it, then your best chance is to try an immediate recovery. Click Delete → Attempt Data Rescue.

This could take a long time, and there is no guarantee of success. *parted* seems to do this faster; see Recipe 8.7.

Discussion

It is usually faster to create a new partition and filesystem, and then replace your files from backup. But it does not hurt to try recovery first.

See Also

- GNOME Partition Editor
- Recipe 8.7

9.7 Resizing Partitions

Problem

You want to make a partition larger or smaller.

Solution

With GParted, this takes just a few clicks. When a partition is resized, the filesystem on it must also be resized. GParted does this in a single operation.

To enlarge a partition, there must be free space at the end of it. Ext4, Btrfs, and XFS can be enlarged online. FAT16/32 must be unmounted first.



Always Have Backups!

Remember, always have current backups!

Figure 9-9 shows a FAT32 filesystem with plenty of free space to grow into.

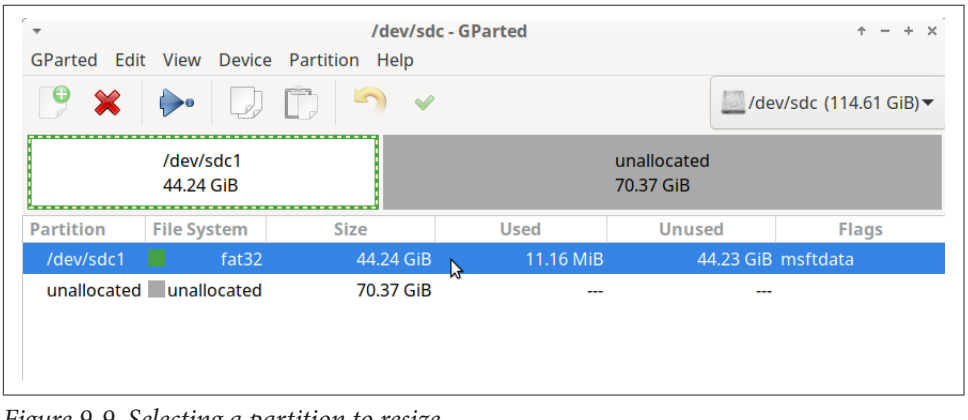


Figure 9-9. Selecting a partition to resize

Right-click on the selected partition to open the menu, then click Resize/Move. This opens a dialog where you set the new size, either by dragging the slider or by typing the value, in mibibytes, in the New Size field (Figure 9-10).

Click Resize/Move, then click the green checkmark. Enlarging a partition takes just a minute or two, and you will see a status message when it is finished.

Use the same procedure for shrinking a partition, except this does not require any free space at the end. Your new partition size should be at least 10% larger than the

space used by your files. Even if you do not plan to add any new files to this filesystem, you must leave a certain amount of unused capacity, because if the filesystem fills up completely, you may not be able to access it. Shrinking a partition takes longer than enlarging it.

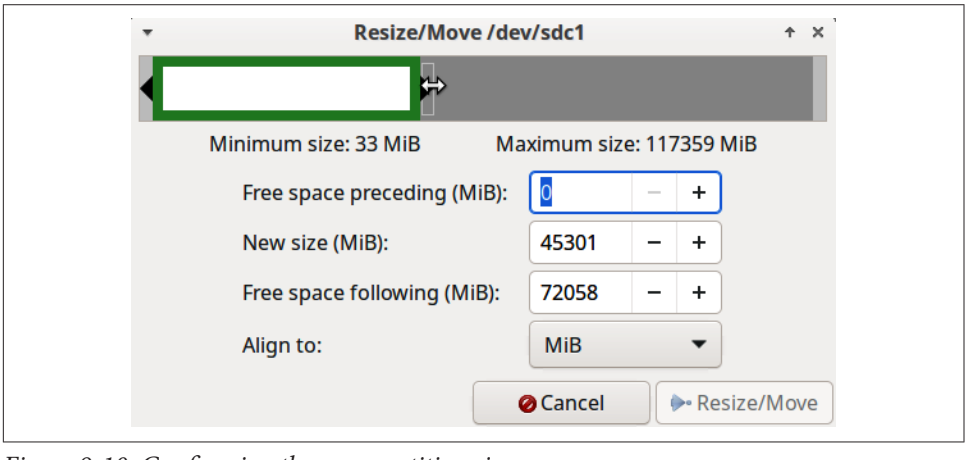


Figure 9-10. Configuring the new partition size

Discussion

The Ext4 filesystem reserves a small amount of space for the root user. If the filesystem fills up, then root can still access the filesystem and remove files. FAT16/32, Btrfs, and XFS do not have reserved blocks.

Ext4 and Btrfs can be shrunk online. XFS can only be enlarged, it cannot be shrunk. It is safer to unmount them before resizing.

See Also

- GNOME Partition Editor
- Recipe 8.8
- Recipe 8.9

9.8 Moving a Partition

Problem

You have a bit of free space between partitions, for example, between */dev/sda1* and */dev/sda2*. You want to move */dev/sda2* into the free space so there is no gap

between them. Or, you wish to enlarge */dev/sda1*, but there is no free space following it, so you have to move */dev/sda2* to make room.

Solution

Right-click the partition you want to move to open the operations menu, then click Resize/Move (Figure 9-11).

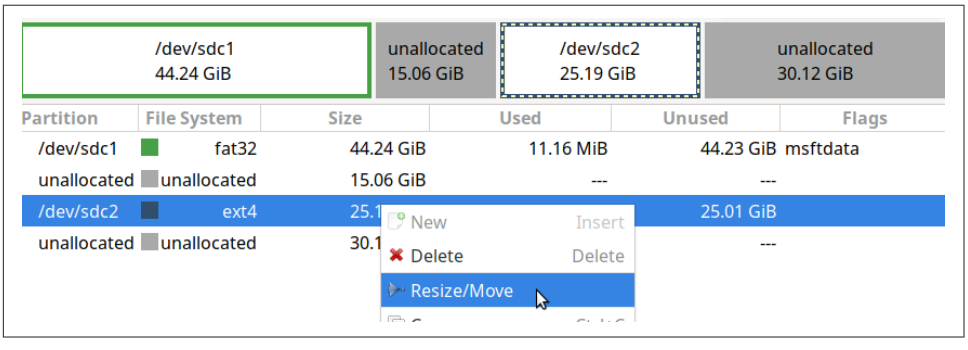


Figure 9-11. Selecting a partition

In the Resize/Move dialog, you can drag the slider to the left, or enter 0 in the Free Space Preceding (MiB) field. Then click Resize/Move (Figure 9-12).

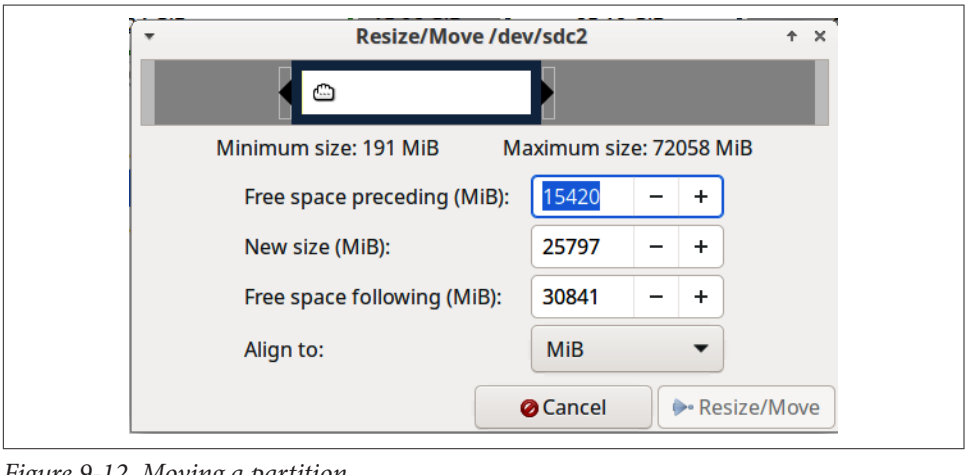


Figure 9-12. Moving a partition

This will take some time, as much as several hours, depending on how much data is on the partition.

Discussion

Moving a partition is more complicated than resizing a partition. When you resize a partition, you are moving only its endpoint, but moving a partition requires also changing its starting point, which, to the operating system, is a big change. GParted usually manages this reliably, but it is risky, so always have good backups.

See Also

- GNOME Partition Editor
- Chapter 19

9.9 Copying a Partition

Problem

You want to make a clone of a partition, or several partitions, as backups or to move data to a new hard disk.

Solution

Use GParted's Copy command. For example, you want to copy */dev/sdb2* to a USB hard drive attached to your system. Copy it to free space equal to or larger than the partition you are copying.

Right-click on the partition you want to copy (Figure 9-13). Unmount it if it is mounted, then click Copy.

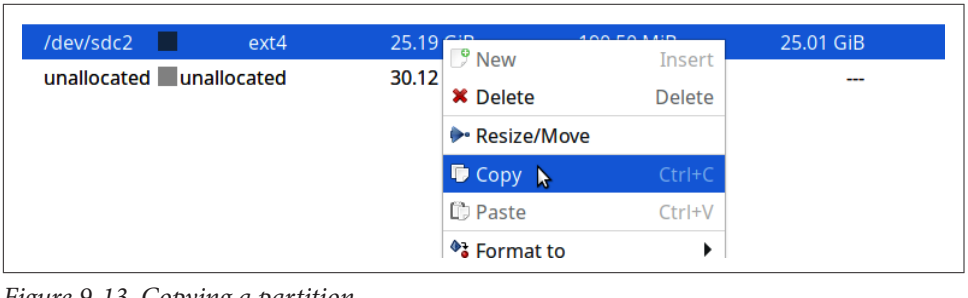


Figure 9-13. Copying a partition

Change to the disk you want to copy it to and click Paste. This opens the configuration dialog, with options to increase the size and change the location of the new partition (Figure 9-14). When you are satisfied with the settings, click Paste.

The final step is to click the green checkmark to start the copy. If you change your mind, click Undo. The copy operation will take a little time, depending on how much data needs to be copied.

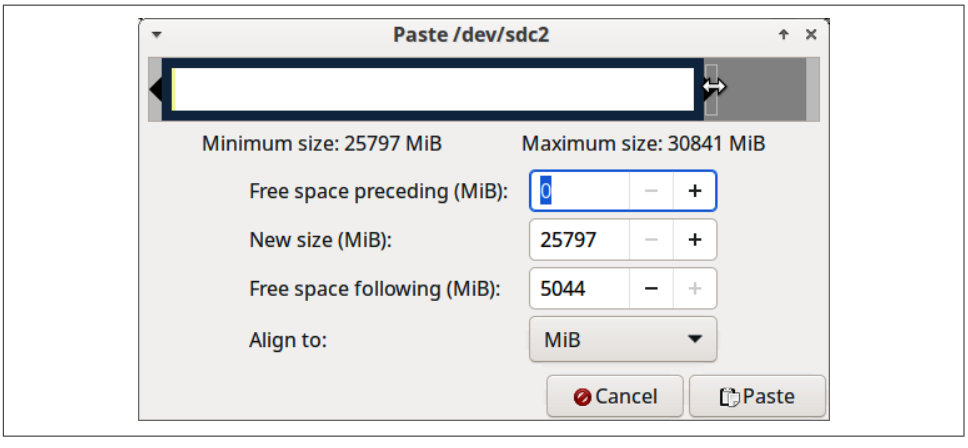


Figure 9-14. Settings for the new partition

Discussion

The copied partition must go into a new partition of equal or greater size. Copying a partition into free space saves the hassle of creating the destination partition.

Copying partitions, in my experience, has limited usefulness. The partition and filesystem UUIDs remain the same, so you cannot use the copied partition on the same system as the original without changing the UUIDs. (Which you can do in GParted in the right-click operations menu.) If you change UUIDs for filesystems listed in */etc/fstab*, their entries must be updated. I think it is better, in most cases, to create new partitions and filesystems, and then copy your files into them.

See Also

- GNOME Partition Editor
- Chapter 11
- Recipe 11.6
- Chapter 19

9.10 Managing Filesystems with GParted

Problem

You want a nice graphical tool for creating new filesystems.

Solution

Use GParted, which manages partitions and filesystems. Find the partition you want to format with a new filesystem, right-click, and select the filesystem you want to use (see Figure 9-15).

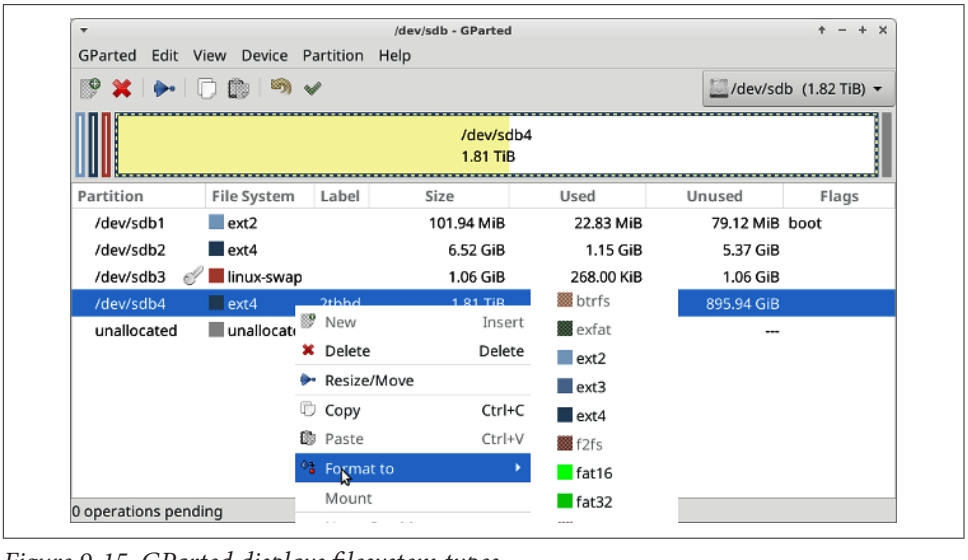


Figure 9-15. GParted displays filesystem types

Click the green checkmark in the toolbar to create the new filesystem. Creating a new filesystem destroys everything on the existing filesystem, so be sure you are in the right place.

Discussion

GParted is one of the best graphical applications in any category. It is a well-organized frontend to a number of command-line tools for managing partitions and filesystems, and it makes complicated tasks simple and fast.

GParted shows the filesystem types on mounted and unmounted volumes, displaying one disk at a time (see Figure 9-16). Click the drop-down menu on the top right to view other disks.

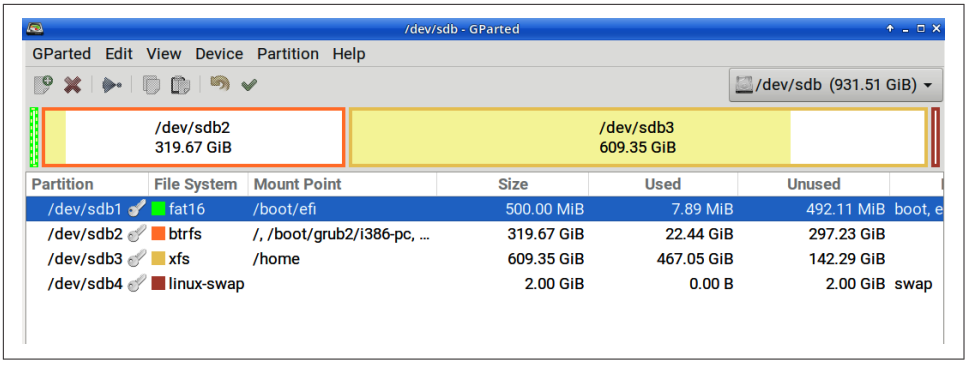


Figure 9-16. GParted displays filesystem types

CHAPTER 10

Getting Detailed Information About Your Computer Hardware

Linux comes with several good utilities for getting detailed information on the hardware components in your computer. You can sit down at a machine and in minutes have an inventory of its components and their specifications, without opening the case.

These utilities are useful for providing detailed information for technical support, finding the correct drivers for a device, and finding out if it is supported in Linux at all. You can't count on manufacturers to provide timely accurate information about their own products. For example, they will often change their chipsets without changing model numbers, which may turn a device that worked fine on Linux into a device that does not work on Linux. Fortunately, in these modern times Linux support is much less of a hassle than it used to be.

Ideally, you will also have your computer documentation, or at least the motherboard manual. Motherboard manuals are usually full of photos, diagrams, and useful information, and you should be able to find them online.

In this chapter you will learn about the *lshw* (list hardware), *lspci* (list PCI), *hwinfo* (hardware information), *lsusb* (list USB), *lscpu* (list CPU), and *lsblk* (list block devices) commands.

lshw and *hwinfo* provide the most complete information.

lshw reports memory configuration, firmware versions, mainboard configuration, CPU version and speed, cache configuration, bus speed, hardware paths, attached devices, partitions, and filesystems.

hwinfo reports computer monitor information, RAID arrays, memory configuration, CPU information, firmware, mainboard configuration, caches, bus speeds, attached devices, partitions, and filesystems.

lsusb probes USB buses and the devices attached to them.

lspci probes PCI buses and the devices attached to them.

lsblk lists physical drives, partitions, and filesystems.

lscpu lists information about your CPU.

10.1 Collecting Hardware Information with lshw

Problem

You want an inventory of the hardware on your system and details about each item.

Solution

Try the *lshw* (Hardware Lister) command with no options, and store the output in a text file:

```
$ sudo lshw | tee hardware.txt
duchess
    description: Laptop
    product: Latitude E7240 (05CA)
    vendor: Dell Inc.
    version: 00
    serial: 456ABC1
    width: 64 bits
[...]
```

You'll get several hundred lines of output that include firmware, drivers, capabilities, serial numbers, version numbers, and bus information. *lshw* will not probe any device attached via a wireless network interface, such as a wireless printer, or a smartphone attached via Bluetooth, but it will report wireless and Bluetooth interfaces.

You may prefer a summary in a hardware path tree view:

```
$ sudo lshw -short
H/W path         Device           Class          Description
============================================================
                                  system         To Be Filled By O.E.M.
/0                                bus            H97M Pro4
/0/0                              memory         64KiB BIOS
/0/b                              memory         16GiB System Memory
/0/b/0                            memory         DIMM [empty]
/0/b/1                            memory         8GiB DIMM DDR3 Synchronous
1333
MHz (0.8 ns)
```

```
[...]
/0/100/14/0/5                    bus           USB3.0 Hub
/0/100/14/0/5/1                  generic       SAMSUNG_Android
/0/100/14/0/5/2                  printer       MFC-J5945DW
/0/100/14/0/5/4  wlx9cefd5fe8f20 network       802.11 n WLAN
/0/100/14/0/b                    input         USB Optical Mouse
/0/100/14/0/c                    input         QuickFire Rapid keyboard
[...]
```

Or try the summary bus view, rather than the hardware path view:

```
$ sudo lshw -businfo
Bus info          Device           Class          Description
============================================================
[...]
cpu@0                              processor      Intel(R) Core(TM) i7-4770K
CPU
@ 3.50GHz
usb@3:5.4         wlx9cefd5fe8f20  network        802.11 n WLAN
usb@3:b                            input          USB Optical Mouse
usb@3:c                            input          QuickFire Rapid keyboard
pci@0000:00:19.0  enp0s25          network        Ethernet Connection (2) I218-V
pci@0000:00:1a.0                   bus            9 Series Chipset Family USB
scsi@0:0.0.0      /dev/sda         disk           4TB ST4000DM000-1F21
scsi@0:0.0.0,1    /dev/sda1        volume         476MiB EXT4 volume
[...]
```

lshw has a graphical interface, which you open with *sudo lshw -X*. This is often a separate package, for example, *lshw-gtk* on Ubuntu and *lshw-gui* on openSUSE and Fedora.

Discussion

lshw packs a lot of information into its output. Visit Hardware Lister (lshw) to learn what everything means.

lshw does not detect FireWire interfaces or computer monitors.

The example says "system To Be Filled By O.E.M." because it is a homebrew machine. A branded computer, such as Lenovo or Dell, should have the brand name and model.

The *H/W path* column contains hardware paths, which are analogous to filepaths. /0 is */system/bus*, which means computer and motherboard. Then all the following entries are in a tree view, similar to a file tree. As you can see in the example output, /0/0 is */system/bus/BIOS memory*, /0/b is the first populated RAM slot, and /0/b/1 is the second populated RAM slot. These paths correspond to physical connections on your motherboard and are commonly called *slots*, even though most of them are soldered to the motherboard and do not have physical slots that you can plug expansion cards into.

See Also

- Hardware Lister (lshw)
- *man 1 lshw*

10.2 Filtering lshw Output

Problem

lshw sure does dump a lot of information, and you want to limit the output to what you want to see.

Solution

Run *sudo lshw -short* or *sudo lshw -businfo* to see a list of device classes, then name one or more device classes that you want to see:

```
$ sudo lshw -short -class bus -class cpu
```

Omit the *-short* option to see detailed information.

Format the long output as HTML, XML, or JSON, and store it in a file so you can use your favorite scripting hacks to parse the output:

```
$ sudo lshw -html -class bus -class cpu | tee lshw.html
$ sudo lshw -xml -class printer -class display -class input | tee lshw.xml
$ sudo lshw -json -class storage | tee lshw.json
```

Remove sensitive information with the *-sanitize* option, such as IP addresses and serial numbers, to make it safer to share with technical support:

```
$ sudo lshw -json -sanitize  -class bus -class cpu | tee lshw.json
```

Discussion

The *tee* command displays output on the screen and stores it in a text file.

See Also

- Hardware Lister (lshw)
- *man 1 lshw*

10.3 Detecting Hardware, Including Displays and RAID Devices, with hwinfo

Problem

You want to get information on your computer monitor and RAID devices, as well as other devices on your system.

Solution

The *hwinfo* command provides a detailed hardware inventory, including monitors and RAID devices on your system. The following example probes your monitor:

```
$ hwinfo --monitor
[...]
  Hardware Class: monitor
  Model: "VIEWSONIC VX2450 SERIES"
  Vendor: VSC "VIEWSONIC"
  Device: eisa 0xe226 "VX2450 SERIES"
[...]
```

The complete output is quite a bit longer than this example, and it includes all supported screen resolutions, date of manufacture, synchronization ranges, type of monitor, and refresh frequencies.

Another nice feature is detecting RAID devices. It does not detect them by default, so use the *--listmd* option:

```
$ hwinfo --listmd
```

If it returns nothing, there are no RAID devices on your system. If there are, it prints a lot of information.

Create a summary of your hardware:

```
$ hwinfo --short
keyboard:
  /dev/input/event4    CM Storm QuickFire Rapid keyboard
mouse:
  /dev/input/event5    CM Storm QuickFire Rapid keyboard
  /dev/input/mice      Logitech Optical Wheel Mouse
printer:
                       Brother Industries MFC-J5945DW
monitor:
                       VIEWSONIC VX2450 SERIES
graphics card:
                       Intel Xeon E3-1200 v3/4th Gen Core Processor Integrated
[...]
```

Get detailed information on one or more hardware components:

```
$ hwinfo --mouse --network --cdrom
```

Consult *man 8 hwinfo* for a list of device names, or run *hwinfo --help*:

```
$ hwinfo --help
Usage: hwinfo [OPTIONS]
Probe for hardware.
Options:
    --<HARDWARE_ITEM>
        This option can be given more than once. Probe for a particular
        HARDWARE_ITEM. Available hardware items are:
        all, arch, bios, block, bluetooth, braille, bridge, camera,
        cdrom, chipcard, cpu, disk, dsl, dvb, fingerprint, floppy,
        framebuffer, gfxcard, hub, ide, isapnp, isdn, joystick, keyboard,
        memory, mmc-ctrl, modem, monitor, mouse, netcard, network, partition,
        pci, pcmcia, pcmcia-ctrl, pppoe, printer, redasd,
        reallyall, scanner, scsi, smp, sound, storage-ctrl, sys, tape,
        tv, uml, usb, usb-ctrl, vbe, wlan, xen, zip
[...]
```

Discussion

hwinfo prints useful and complete information. For example, for network interfaces it shows their */sys* paths, drivers, link status, and MAC addresses. CD-ROM output includes the model name, revision number, drivers, device files, drive speed, a features list, and whether there is a disk in the drive. *hwinfo* often tells you more than the manufacturer's product information.

See Also

- *man 8 hwinfo*
- hwinfo on GitHub

10.4 Detecting PCI Hardware with lspci

Problem

You want to list devices attached to the PCI bus on your computer with vendor and version information.

Solution

Run the *lspci* (list PCI) command. The following example prints a summary list of all PCI devices:

```
$ lspci
00:00.0 Host bridge: Intel Corporation 4th Gen Core Processor DRAM Controller
(rev 06)
00:02.0 VGA compatible controller: Intel Corporation Xeon E3-1200 v3/4th Gen
Core Processor Integrated Graphics Controller (rev 06)
00:03.0 Audio device: Intel Corporation Xeon E3-1200 v3/4th Gen Core Processor
HD Audio Controller (rev 06)
[...]
```

Increase verbosity to see more details:

```
$ lspci -v
$ lspci -vv
$ lspci -vvv
```

When you see "access denied" messages, try *sudo lspci* to see what you're missing.

Discussion

lspci reads information from the PCI bus, which includes onboard components on your motherboard as well as expansion cards plugged into PCI slots.

lspci displays additional information from its own database of hardware IDs, such as vendors, devices, and classes and subclasses. This information is stored in a text file in various locations, depending on your Linux distribution. Ubuntu puts it in */usr/share/misc/pci.ids*, Fedora uses */usr/share/hwdata/pci.ids*, and openSUSE uses */usr/share/pci.ids*. The man page for your Linux should tell you where it is, or search for the *pci.ids* file (*locate pci.ids*).

The *lspci* maintainers welcome submissions of updated information; read your *pci.ids* file for instructions. Run the *sudo update-pciids* command periodically to update your PCI IDs database.

PCI is short for Peripheral Component Interconnect. PCI is a local hardware bus; that is, a means for the various hardware devices in your computer to communicate with the Linux kernel. *lspci* primarily detects controllers, buses, and some individual devices, including:

- SATA controllers
- Audio controllers and devices
- Video controllers and devices
- Ethernet controllers
- USB controllers
- Communication controllers
- Ethernet controllers
- RAID controllers

- Integrated SD/MMC card readers
- PCI FireWire controllers

There have been several PCI protocols over the years. The current standard is PCIe, PCI Express, introduced in 2003. It is backward compatible with all legacy PCI protocols, and it replaces PCI, PCI-X, and AGP. Remember AGP, the Accelerated Graphics Port protocol? AGP video cards were faster than PCI video cards because AGP provided a dedicated link for video processing.

PCIe is substantially different from the earlier protocols because, just like AGP, each device gets it own dedicated link. The older protocols used a shared parallel bus, which was considerably slower.

See Also

- *man 8 lspci*
- *man 8 update-pciids*

10.5 Understanding lspci Output

Problem

Most of the output of *lspci* makes sense because it is device specifications. But you want to know what the numbers at the beginning of each device line are for, like this example:

```
$ lspci
[...]
00:1f.2 SATA controller: Intel Corporation 9 Series Chipset Family SATA
Controller [AHCI Mode]
[...]
```

Solution

00:1f.2 is the device's BDF number, *bus:device.function*. Bus number 00, device number 1f, and function number 2. A function number of 2 means the device has two functions, and each one gets its own PCI address.

Use the tree view to see the relationship between the PCI bus and the devices:

```
$ lspci -tvv
-[0000:00]-+-00.0  Intel Corporation 4th Gen Core Processor DRAM Controller
           +-02.0  Intel Corporation Xeon E3-1200 v3/4th Gen Core Processor
                   Integrated Graphics Controller
           +-03.0  Intel Corporation Xeon E3-1200 v3/4th Gen Core Processor HD
                   Audio Controller
```

```
+-14.0  Intel Corporation 9 Series Chipset Family USB xHCI Controller
+-16.0  Intel Corporation 9 Series Chipset Family ME Interface #1
+-19.0  Intel Corporation Ethernet Connection (2) I218-V
+-1a.0  Intel Corporation 9 Series Chipset Family USB EHCI
        Controller #2
+-1b.0  Intel Corporation 9 Series Chipset Family HD Audio Controller
+-1c.0-[01]--
+-1c.3-[02-03]----00.0-[03]--
+-1d.0  Intel Corporation 9 Series Chipset Family USB EHCI
        Controller #1
+-1f.0  Intel Corporation H97 Chipset LPC Controller
+-1f.2  Intel Corporation 9 Series Chipset Family SATA Controller
        [AHCI Mode]
\-1f.3  Intel Corporation 9 Series Chipset Family SMBus Controller
```

PCs almost always have a single PCI bus, which is always 00.

Discussion

The zeroes enclosed in brackets at the root of the tree, [0000:00], identify the *domain* and *bus*. The first four zeroes are the domain number, and the two zeroes after the colon are the bus number. The domain is the host bridge. The PCI host bridge connects the PCI controller to the CPU. *Domain* is a Linux-specific term, and it is more commonly called the *segment group*. You can also see this with the *-D* option:

```
$ lspci -D
0000:00:00.0 Host bridge: Intel Corporation 4th Gen Core Processor DRAM
 Controller (rev 06)
0000:00:02.0 VGA compatible controller: Intel Corporation Xeon E3-1200 v3/4th Gen
 Core Processor Integrated Graphics Controller (rev 06)
0000:00:03.0 Audio device: Intel Corporation Xeon E3-1200 v3/4th Gen Core
 Processor HD Audio Controller
[...]
```

You will see multiple host bridges on servers with multiple physical CPUs, and sometimes multiple buses on a single domain.

See Also

- *man 8 lspci*

10.6 Filtering lspci Output

Problem

lspci outputs a lot of information, and you want to filter it to see just what you want to see.

Solution

Use the *awk* command to cut out the clutter. The following example finds only entries pertaining to USB:

```
$ lspci -v | awk '/USB/,/^$/'
00:14.0 USB controller: Intel Corporation 9 Series Chipset Family USB xHCI
Controller (prog-if 30 [XHCI])
        Subsystem: ASRock Incorporation 9 Series Chipset Family USB xHCI
Controller
        Flags: bus master, medium devsel, latency 0, IRQ 26
        Memory at efc20000 (64-bit, non-prefetchable) [size=64K]
        Capabilities: <access denied>
        Kernel driver in use: xhci_hcd

00:1a.0 USB controller: Intel Corporation 9 Series Chipset Family USB EHCI
Controller #2 (prog-if 20 [EHCI])
        Subsystem: ASRock Incorporation 9 Series Chipset Family USB EHCI
Controller
        Flags: bus master, medium devsel, latency 0, IRQ 16
        Memory at efc3b000 (32-bit, non-prefetchable) [size=1K]
        Capabilities: <access denied>
        Kernel driver in use: ehci-pci
```

You must use the classes (Audio, Ethernet, USB, etc.) as they appear in the output from *lspci*, and pay attention to case, because using *awk* to run a case-insensitive search is complicated. This example shows the audio controller and device:

```
$ lspci -v | awk '/Audio/,/^$/'
00:03.0 Audio device: Intel Corporation Xeon E3-1200 v3/4th Gen Core Processor
HD Audio Controller (rev 06)
        Subsystem: ASRock Incorporation Xeon E3-1200 v3/4th Gen Core Processor
HD Audio Controller
        Flags: bus master, fast devsel, latency 0, IRQ 31
        Memory at efc34000 (64-bit, non-prefetchable) [size=16K]
        Capabilities: <access denied>
        Kernel driver in use: snd_hda_intel
        Kernel modules: snd_hda_intel

00:1b.0 Audio device: Intel Corporation 9 Series Chipset Family HD Audio
Controller
        Subsystem: ASRock Incorporation 9 Series Chipset Family HD Audio
Controller
        Flags: bus master, fast devsel, latency 0, IRQ 32
        Memory at efc30000 (64-bit, non-prefetchable) [size=16K]
        Capabilities: <access denied>
        Kernel driver in use: snd_hda_intel
        Kernel modules: snd_hda_intel
```

Adjust the verbosity level as needed.

You may also select items by vendor, device, or class number. Find these numbers with the *-nn* option. In this example, `0300` (enclosed in square braces) is the class number, `8086` is the vendor number, and `0412` is the device number:

```
$ lspci -nn
[....]
00:02.0 VGA compatible controller [0300]: Intel Corporation
Xeon E3-1200 v3/4th Gen Core Processor Integrated Graphics Controller
[8086:0412] (rev 06)
[...]
```

The following examples filter by class, vendor, and device, respectively:

```
$ lspci -d ::0604
00:1c.0 PCI bridge: Intel Corporation 9 Series Chipset Family PCI Express Root
Port 1 (rev d0)
00:1c.3 PCI bridge: Intel Corporation 82801 PCI Bridge (rev d0)
02:00.0 PCI bridge: ASMedia Technology Inc. ASM1083/1085 PCIe to PCI Bridge (rev
03)

$ lspci -d 8086::
00:00.0 Host bridge: Intel Corporation 4th Gen Core Processor DRAM Controller
(rev 06)
00:02.0 VGA compatible controller: Intel Corporation Xeon E3-1200 v3/4th Gen
Core Processor Integrated Graphics Controller (rev 06)
00:03.0 Audio device: Intel Corporation Xeon E3-1200 v3/4th Gen Core Processor
HD Audio Controller (rev 06)
[...]

$ lspci -d :0412:
00:02.0 VGA compatible controller: Intel Corporation Xeon E3-1200 v3/4th Gen
Core Processor Integrated Graphics Controller (rev 06)
```

Another way to find these numbers is to look them up at the PCI ID Repository.

Discussion

awk is a marvelous power tool for extracting specific text strings from command output or documents. The caret, ^, is a regular expression anchor that matches the start of a string, and *$* matches the end, so in this example, /^$/ looks for the line breaks, the empty spaces at the beginning and end of the text blocks. This is a great trick for extracting text blocks from sources that have spaces between sections.

See Also

- *man 1 grep*
- *man 8 lspci*
- the PCI ID Repository

10.7 Using lspci to Identify Kernel Modules

Problem

You want to know which kernel modules your PCI devices are using, and which ones are available on your system.

Solution

Use the *-k* option. The following example queries only the Ethernet controller:

```
$ lspci -kd ::0200
00:19.0 Ethernet controller: Intel Corporation Ethernet Connection (2) I218-V
        Subsystem: ASRock Incorporation Ethernet Connection (2) I218-V
        Kernel driver in use: e1000e
        Kernel modules: e1000e
```

You may also use *awk*, like this example for your graphics controller:

```
$ lspci -vmmk| awk '/VGA/,/^$/'
Class:  VGA compatible controller
Vendor: Intel Corporation
Device: Xeon E3-1200 v3/4th Gen Core Processor Integrated Graphics Controller
SVendor:        ASRock Incorporation
SDevice:        Xeon E3-1200 v3/4th Gen Core Processor Integrated Graphics
Controller
Rev:    06
Driver: i915
Module: i915
```

Discussion

The *-k* option shows the kernel modules in use, and all of the available kernel modules for each device. Usually the in-use and available entries are the same, but sometimes there are multiple modules available.

When you use *awk* remember to add some verbosity, or you may not see the information you want. See the Discussion in Recipe 10.6 to learn about the *awk* options.

See Also

- *man 1 awk*
- *man 8 lspci*

10.8 Using lsusb to List USB Devices

Problem

You want a quick, easy tool for listing USB devices on your system.

Solution

lsusb lists USB buses and connected USB devices, including mice, keyboards, USB sticks, printers, smartphones, and other connected peripherals. The following two examples show two different views of the same devices.

Run *lsusb* with no options to see a summary of USB devices on your system. In the next example, three external USB devices are connected: a keyboard, mouse, and wireless network interface:

```
$ lsusb
[...]
Bus 003 Device 011: ID 148f:5372 Ralink Technology, Corp. RT5372 Wireless Adapter
Bus 003 Device 002: ID 0bda:5401 Realtek Semiconductor Corp. RTL 8153 USB 3.0
 hub with gigabit ethernet
Bus 003 Device 006: ID 046d:c018 Logitech, Inc. Optical Wheel Mouse
Bus 003 Device 005: ID 2516:0004 Cooler Master Co., Ltd. Storm QuickFire Rapid
 Mechanical Keyboard
[...]
```

This example shows the same thing, with more details in a USB bus hierarchy format, including kernel drivers, device codes and vendor numbers, and port numbers:

```
$ lsusb -tv
[...]
/:  Bus 03.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/14p, 480M
    ID 1d6b:0002 Linux Foundation 2.0 root hub
    |__ Port 3: Dev 2, If 0, Class=Hub, Driver=hub/4p, 480M
        ID 0bda:5401 Realtek Semiconductor Corp. RTL 8153 USB 3.0 hub with
        gigabit ethernet
    |__ Port 7: Dev 11, If 0, Class=Vendor Specific Class, Driver=rt2800usb, 480M
        ID 148f:5372 Ralink Technology, Corp. RT5372 Wireless Adapter
    |__ Port 11: Dev 5, If 0, Class=Human Interface Device, Driver=usbhid, 1.5M
        ID 2516:0004 Cooler Master Co., Ltd. Storm QuickFire Rapid Mechanical
        Keyboard
    |__ Port 12: Dev 6, If 0, Class=Human Interface Device, Driver=usbhid, 1.5M
        ID 046d:c018 Logitech, Inc. Optical Wheel Mouse
[...]
```

The following examples show what it looks like when you plug in an external USB hub with a Bluetooth interface and a Samsung smartphone attached to the hub:

```
$ lsusb
[...]
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

```
Bus 003 Device 012: ID 04e8:6860 Samsung Electronics Co., Ltd Galaxy series,
 misc. (MTP mode)
Bus 003 Device 013: ID 0a12:0001 Cambridge Silicon Radio, Ltd Bluetooth Dongle
 (HCI mode)
Bus 003 Device 002: ID 0bda:5401 Realtek Semiconductor Corp. RTL 8153 USB 3.0
 hub with gigabit ethernet
[...]

$ lsusb -tv
[...]
/:  Bus 03.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/14p, 480M
    ID 1d6b:0002 Linux Foundation 2.0 root hub
    |__ Port 3: Dev 2, If 0, Class=Hub, Driver=hub/4p, 480M
        ID 0bda:5401 Realtek Semiconductor Corp. RTL 8153 USB 3.0 hub with
        gigabit ethernet
        |__ Port 4: Dev 12, If 0, Class=Imaging, Driver=, 480M
            ID 04e8:6860 Samsung Electronics Co., Ltd Galaxy series, misc. (MTP
            mode)
        |__ Port 2: Dev 13, If 0, Class=Wireless, Driver=btusb, 12M
            ID 0a12:0001 Cambridge Silicon Radio, Ltd Bluetooth Dongle (HCI mode)
        |__ Port 2: Dev 13, If 1, Class=Wireless, Driver=btusb, 12M
            ID 0a12:0001 Cambridge Silicon Radio, Ltd Bluetooth Dongle (HCI mode)
[...]
```

Discussion

The bus and port numbers are always the same. The dev number changes every time you plug in a device.

The ID numbers, for example, `0a12:0001`, are the vendor and device codes. Manufacturers must apply to *https://usb.org* for new codes. You can find the list of current USB IDs at linux-usb.org and to contribute updated information.

The class codes are also managed by *https://usb.org*; see USB class codes. I find it interesting that Dev 57, which is a Samsung Android phone, is classed as an imaging device. However, it makes sense because most Linux distributions use the Media Transfer Protocol (MTP) to transfer files from an Android phone.

The examples in this section are from a PC that has both USB 2.0 and USB 3.1 ports. The *lsusb* output shows the negotiated speeds the devices are using, so when you see something like `usbhid, 1.5M`, instead of `480M` or `5000M`, that is all right because that is a keyboard, which does not need the full speed of the USB link. You should see higher speeds for storage devices, such as USB sticks and external hard drives.

See Also

- *man 8 lsusb*
- *https://usb.org*
- *https://oreil.ly/js1oj*

10.9 Listing Partitions and Hard Disks with lsblk

Problem

You need a quick way to list all of your attached storage drives, and their partitions.

Solution

Use the *lsblk* (list block devices) command. Run it with no options to generate a list of all block devices on your computer:

```
$ lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0   3.7T  0 disk
├─sda1   8:1    0   476M  0 part /boot
├─sda2   8:2    0  55.9G  0 part /
├─sda3   8:3    0   1.8T  0 part /home
└─sda4   8:4    0   7.5G  0 part [SWAP]
sdb      8:16   0   1.8T  0 disk
├─sdb1   8:17   0   102M  0 part
├─sdb2   8:18   0   6.5G  0 part
├─sdb3   8:19   0   1.1G  0 part [SWAP]
└─sdb4   8:20   0   1.8T  0 part
sdc      8:32   0   3.7T  0 disk
├─sdc1   8:33   0   128M  0 part
├─sdc2   8:34   0 439.7G  0 part
└─sdc3   8:35   0   3.2T  0 part
sdd      8:48   1   3.8G  0 disk
└─sdd1   8:49   1   3.8G  0 part
sr0     11:0    1 159.3M  0 rom
```

Show the filesystem labels and UUIDs on the selected device:

```
$ lsblk -f /dev/sdc
NAME   FSTYPE LABEL                UUID                 MOUNTPOINT
sdc
├─sdc1
├─sdc2 ntfs   Seagate Backup Plus   2E203F82203F5057
└─sdc3 ext4   backup                0451d428-9716-4cdd  /media/max/backup
```

List only SCSI devices and their types:

```
$ lsblk -S
NAME HCTL       TYPE VENDOR   MODEL             REV TRAN
sda  0:0:0:0    disk ATA      ST4000DM000-1F21 CC54 sata
sdb  2:0:0:0    disk ATA      SAMSUNG HD204UI  0001 sata
sdc  6:0:0:0    disk Seagate  BUP SL           0304 usb
sr0  4:0:0:0    rom  ATAPI    iHAS424   B      GL1B sata
```

Discussion

sda and *sdb* are SATA hard disks, and *sdc* is a USB flash drive. On Linux, mass storage devices such as SATA hard disks and flash media use the SCSI driver. *sr0*, *rom*, and *ATAPI* all identify a CD/DVD player.

Defining the term *block devices* without starting arguments is rather difficult because it's a programming term that does not translate well to a concise userland concept. In my experience it is most useful to think of block devices as mass storage devices and the partitions on storage devices.

`MAJ:MIN` are the major and minor numbers. The major number identifies the category, for example, 8 is for *sd* devices, and the minor number labels each device in sequence. (Run **`lsblk -l`** to see this in a tree structure.)

`RM` tells if it a removable drive or not, with 1 indicating a removable drive.

`SIZE` is the size of the block device.

`RO = 0` means the device is not read-only, and 1 is read-only. *sr0*, the CD/DVD drive, is a read-write drive, but *lsblk* cannot tell you if the disk in *sr0* is writeable.

`TYPE` identifies the disk type.

`MOUNTPOINT` shows the paths, if the device is mounted.

See Also

- *man 8 lsblk*

10.10 Getting CPU Information

Problem

You want to know what CPU or CPUs are on your system, and their specifications.

Solution

Run the *lscpu* (list CPU) command with no options:

```
$ lscpu
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              8
On-line CPU(s) list: 0-7
Thread(s) per core:  2
Core(s) per socket:  4
Socket(s):           1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               60
Model name:          Intel(R) Core(TM) i7-4770K CPU @ 3.50GHz
[...]
L1d cache:           128 KiB
L1i cache:           128 KiB
L2 cache:            1 MiB
L3 cache:            8 MiB
[...]
```

This spits out a large amount of information; you will also see a large number of flags, which list capabilities, and L cache information.

Discussion

There are three types of CPU caches: L1, L2, and L3. These are small memory caches on the CPU. They are very fast, many times faster than system RAM, and store the data that the CPU is most likely to need for its next operations. L1 is the fastest and most expensive, so it is usually the smallest. L2 is the next fastest and less expensive, and is usually larger than L1. L3 is the slowest and least expensive, and usually the largest.

The CPU in the preceding example has four caches. Use the *-C* option to see more detailed cache information:

```
$ lscpu -C
NAME ONE-SIZE ALL-SIZE WAYS TYPE        LEVEL
L1d       32K     128K    8 Data            1
L1i       32K     128K    8 Instruction     1
L2       256K       1M    8 Unified         2
L3         8M       8M   16 Unified         3
```

This shows four caches, shared among the four physical CPU cores. L1i caches store CPU instructions, and L1d caches store data. L2 and L3 store data.

The number of CPU cores can be a little confusing. `CPU(s): 8` does not mean 8 physical cores in this example; instead, that is how many cores the Linux kernel sees. The following lines tell the full story:

```
Thread(s) per core:  2
Core(s) per socket:  4
Socket(s):           1
```

This is a single processor with four physical cores and two threads per core, for a total of eight logical CPUs.

See Also

- *man 1 lscpu*

10.11 Identifying Your Hardware Architecture

Problem

You're not sure what the hardware architecture is on a machine; you think it is either x86-64 or ARM, and you need to know which it is.

Solution

Use the *uname* command. This example is on an x86-64 machine:

```
$ uname -m
x86_64
```

The following list contains some of the more common results you might see:

- arm
- aarch64
- armv7* (arm7 and below are 32-bit)
- armv8* (arm8 and up is 64-bit)
- ia64
- ppc
- ppc64
- s390x
- sparc
- sparc64
- i386

- i686
- x86_64

If the machine is not running Linux, try booting it with a SystemRescue USB stick, and then run *uname -m*.

You can install Linux on a Chromebook. Chromebooks use both Intel and ARM processors. One way to see what yours has is to open the web browser to *chrome://system*. This shows all the system information, probably more than you want.

A friendlier tool is Cog System Info Viewer, which displays hardware and network information on Chromebooks.

Discussion

Linux supports more hardware architectures than any other operating system, from tiny embedded systems and systems on a chip (SoCs), to mainframes and supercomputers and everything in between. Whatever obscure computing hardware you might have, chances are that some flavor of Linux will run on it.

See Also

- *man 1 uname*

CHAPTER 11

Creating and Managing Filesystems

Linux supports a lot of filesystems, more than any other operating system. Filesystems are essential to computing and do an astounding amount of work. A computer filesystem stores, organizes, and protects our data, and is under continual stress from being constantly in use. As Linux users we are fortunate to have many first-rate filesystems to choose from.

In this chapter you will learn about the command-line tools for creating and managing the following general-purpose filesystems, which are fully supported on Linux and well maintained:

- Ext4, the Extended Filesystem
- XFS, the X File System; the X stands only for X
- Btrfs, the b-tree filesystem, pronounced Butter FS
- FAT16/32, File Allocation Table 16- and 32-bit
- exFAT, Extended FAT, Microsoft's newest 64-bit filesytem

Not included in this chapter are Microsoft's NTFS or Apple's HFS/HFS+/APFS. Linux has good support for Microsoft's NTFS, both read and write. To try it, look for *ntfs-3g* (NTFS third generation) packages.

Support for Apple's HFS/HFS+/APFS is unreliable. To give it a test drive, look for packages with *hfs* or *apfs* in the names, and make sure the description specifies they are for Apple filesystems.

There are many special-purpose filesystems, such as UBIFS and JFFS2 for CompactFlash devices; the compressed filesystem SquashFS, HDFS, CephFS, and GlusterFS for distributed computing; NFS for network file sharing; and many more. These

could easily fill a large book by themselves and are not included here. They are freely available to try out and learn.

Filesystem Overview

Before you can use any storage device, such as a hard disk, USB flash drive, or SD card, it must be partitioned and formatted with a filesystem. Every filesystem must have its own disk partition. A partition can cover an entire disk, or a disk can be divided into multiple partitions. Each partition is like an independent disk, and each partition can have a different filesystem.

A filesystem must be mounted, or attached, to the running filesystem before it is accessible. A filesystem needs a *mountpoint*, which is a directory created for that filesystem. This directory can be anywhere, though the traditional locations are */mnt* and */media*.

You may mount only one filesystem per mountpoint. If you mount a second filesystem, it overwrites the first filesystem.

A filesystem can be set up to mount automatically at system startup, dynamically when you attach removable media, manually from the command line, or by clicking a button on your desktop or in your file manager. Most Linux distributions take good care of handling removable media. Plug in your USB device or optical disk, and Linux takes care of setting up the mountpoint and automounting it, or setting it up for you to mount it with the click of a button (Figure 11-1).

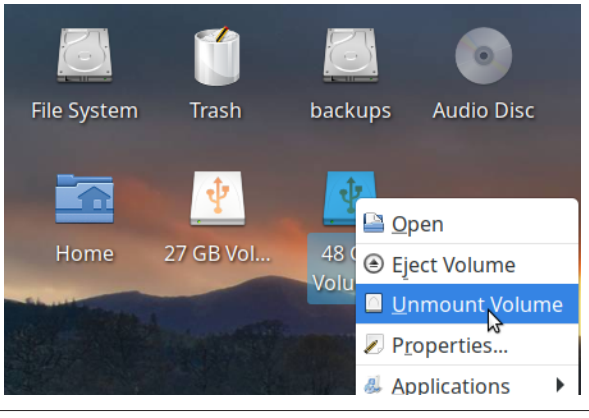


Figure 11-1. Removable media buttons on Xfce desktop

Ext4, XFS, Btrfs, and exFAT are *64-bit* filesystems. This means they support a 64-bit block addressing space, which enables much larger file and filesystem sizes than 32- and 16-bit filesystems. 64-bit computing has been around at least since the 1970s on

supercomputers, then later on high-end business machines like IBM Power and Sun Microsystems UltraSPARC.

My first Windows 3.1/DOS PC, back in the mid-1990s, was a 16-bit system. Windows 95 boasted of being the first 32-bit consumer operating system. The first 64-bit filesystems for x86 PCs started appearing in Linux around 2001. See Ext4 High Level Design in the Linux kernel documentation to see nice tables that lay this all out for us, comparing 32- and 64-bit filesystems.

64-bit filesystems are backward compatible with 32-bit applications. After all these years it is unlikely you will run into 32-bit apps, though if you do they will run on your modern Linux, provided that it supplies the necessary packages to set up a 32-bit environment.

Ext4 and XFS are *journaling* filesystems, and Btrfs is a *copy-on-write* (CoW) filesystem. Journaling and CoW keep your filesystems in consistent states even after a power failure or system crash. Filesystems are complex and busy, and an interruption affects more than just the files you are working on. Interruptions result in large numbers of files with incompleted tasks, and in the olden days this meant possibly losing your whole filesystem.

Ext4 is the most widely used filesystem on Linux and is the default on the majority of Linux distributions. It's not exciting. It's well tested, well supported, and does its job without drama. The Ext4 journal records changes until they are written to disk, providing protection from data loss in the event of an interruption. Ext4 filesystems can be resized, both larger and smaller.

XFS was originally a high-performance Unix 64-bit filesystem, ported to Linux in 2001. XFS is a fast, efficient, reliable journaling filesystem suitable for systems from small personal machines and to multidisk datacenter setups. XFS can be resized larger, but not smaller.

Btrfs is an advanced copy-on-write (CoW) filesystem that includes a batch of features not present in the other filesystems in this chapter, including snapshots; RAID 0, 1, and 10; and subvolumes. Subvolumes are wonderfully flexible, as they enable creating multiple filesystem roots on a single partition. CoW is a cool way of creating snapshots in a space-efficient way, where each snapshot contains only the changes from the previous snapshot. When you run into problems, you can roll back to an older known good snapshot. Btrfs resizes smaller and larger.

FAT16/32 are the elderly Microsoft 16- and 32-bit filesystems. FAT32 is the most universal filesystem, supported by Microsoft Windows, Apple's macOS, Linux, Unix, and DOS operating systems. Use FAT32 for easiest file sharing on portable media. It has one limitation that is a showstopper for some uses, and that is a maximum file size of 4 GB (on media with 4K blocks).

exFAT is the newest Microsoft 64-bit filesystem, a nice upgrade from FAT32. exFAT is a fast, lightweight filesystem for USB sticks and SD media, and supports much larger file and volume sizes than FAT32. Wikipedia cites a 16 EiB maximum file size and 128 PiB maximum volume size. It does not have a journal or CoW.

exFAT is troublesome for Linux users because it is a patented proprietary filesystem, which was not available to Linux as a native filesystem until 2020. You need to worry about Linux compatibility only if you want to read and copy USB flash drives or SDXC cards formatted with exFAT to your Linux computer. For example, you want to use exFAT-formatted SDXC cards with your digital camera, or audio recording device.

To use exFAT with Linux you have two options. One is to use the *exfatprogs*, or *exfat-fuse* and *exfat-utils* packages, which are available on most distributions. exFAT FUSE was developed and is maintained outside of the US, making it immune to US patent laws. exFAT FUSE takes advantage of Filesystem in Userspace (FUSE), which enables unprivileged users to run filesystems in userspace. It is not as efficient as a filesystem properly integrated into the kernel, but it works, and you can read and write exFAT files. Some hardy souls try to use exFAT FUSE in shared partitions to share files with Windows and macOS. In theory this should work, though there are sometimes glitches related to how well a particular Windows or macOS release implements exFAT.

The other option is to wait a little while for native support. Microsoft released exFAT in 2006 and licensed it primarily to companies that make embedded systems and embedded media. But times change. Microsoft has become an open source contributor, and a member of the Open Invention Network (OIN). Microsoft released the exFAT specification in 2019. Releasing the specification sidestepped licensing hassles with the existing exFAT code, and Linux kernel developers wasted no time writing new code. Native support for exFAT with this shiny new code is in Linux kernel 5.7. This should find its way into your favorite distro soon; run *uname -r* to see your kernel version.

11.1 Listing Supported Filesystems

Problem

You need to know what filesystems are installed on your Linux system.

Solution

Read */proc/filesystems* to see a list of installed filesystems:

```
$ cat /proc/filesystems
nodev   sysfs
```

```
nodev   tmpfs
nodev   bdev
nodev   proc
nodev   cgroup
nodev   cgroup2
nodev   cpuset
nodev   devtmpfs
nodev   debugfs
nodev   tracefs
nodev   securityfs
nodev   sockfs
nodev   bpf
nodev   pipefs
nodev   ramfs
nodev   hugetlbfs
nodev   devpts
        ext3
        ext2
        ext4
nodev   autofs
nodev   mqueue
nodev   pstore
        btrfs
        vfat
        xfs
        fuseblk
nodev   fuse
nodev   fusectl
        jfs
        nilfs2
```

Discussion

See all those *nodev* entries? Those are all virtual filesystems that exist only in memory and are not attached to a physical device like */dev/sda1*. Systemd manages all of these virtual filesystems.

The other filesystems, Ext4, XFS, and so on, are the filesystems we use on our storage devices to store, organize, and protect our data.

See Also

- "sysfs, the filesystem for exporting kernel objects" is written for developers, but it has useful information for Linux users and admins.

11.2 Identifying Your Existing Filesystems

Problem

You do not know what filesystems are already on your system, or on a removable storage disk, and you need to know how to list them.

Solution

Use the *lsblk* command. You can list just the device names and filesystems with the *NAME* and *FSTYPE* options:

```
$ lsblk -o NAME,FSTYPE
NAME   FSTYPE
sda
├─sda1 vfat
├─sda2 btrfs
├─sda3 xfs
└─sda4 swap
sdb
├─sdb1 ext2
├─sdb2 ext4
├─sdb3 swap
└─sdb4 LVM2_member
sdc
└─sdc1 vfat
sr0
```

Query a single disk:

```
$ lsblk -o NAME,FSTYPE /dev/sdb
├─sdb1 ext2
├─sdb2 ext4
├─sdb3 swap
└─sdb4 LVM2_member
```

Or a single partition:

```
$ lsblk -o NAME,FSTYPE /dev/sda1
NAME FSTYPE
sda1 vfat
```

This is my favorite *lsblk* incantation. It shows all device names, filesystem types, filesystem sizes, percentage used, labels, and mountpoints:

```
$ lsblk -o NAME,FSTYPE,LABEL,FSSIZE,FSUSE%,MOUNTPOINT
NAME   FSTYPE   LABEL     FSSIZE FSUSE% MOUNTPOINT
loop0  squashfs           646.5M   100% /run/archiso/sfs/airootfs
sda
├─sda1
└─sda2 ntfs
sdb
```

```
├─sdb1 vfat     BOOT
├─sdb2 btrfs    root
├─sdb3 xfs      home
└─sdb4 swap
sdc    iso9660  RESCUE800
└─sdc1 iso9660  RESCUE800   708M   100% /run/archiso/bootmnt
sr0
```

Discussion

Run *lsblk --help* to see a list of columns. There is quite a bit of useful information, such as PATH, LABEL, UUID, HOTPLUG, MODEL, SERIAL, and SIZE.

On some distros you may need root permissions to see the filesystem types, UUIDs, and labels.

lsblk always prints *vfat* for both FAT16 and FAT32 filesystems. Use GParted or *parted* to see whether a filesystem is FAT16 or FAT32.

vfat is Virtual FAT, the kernel's filesystem driver for FAT16 and FAT32.

See Also

- Linux Kernel SCSI Interfaces Guide
- Major and minor numbers for block and character devices
- *man 8 lsblk*
- *man 8 parted*
- Chapter 8
- Chapter 9

11.3 Resizing Filesystems

Problem

You want to enlarge or reduce the size of your filesystem.

Solution

Every filesystem has its own commands for resizing. See Recipes 8.8, 8.9, and 9.7 to learn about resizing filesystems.

Discussion

The filesystem's partition must also be resized to match. GParted does this in a single operation (see Recipe 9.7).

Recipes 8.8 and 8.9 use *parted* and filesystem utilities to resize a filesystem and its partition in two steps.

See Also

- Recipe 8.8
- Recipe 8.9
- Recipe 9.7
- *man 8 resize2fs*
- *man 8 parted*
- *man 8 xfs_growfs*
- *man 8 btrfs*
- *man 8 fsck.vfat*

11.4 Deleting Filesystems

Problem

You need to delete a filesystem and its underlying partition.

Solution

To delete the filesystem and its partition, use *parted*. In this example, */dev/sdb1* is deleted. Verify which partition and filesystem you are going to delete, then make sure the filesystem is unmounted. In the example, the mountpoint is */media/duchess/stuff*:

```
$ lsblk -f
sda
├─sdb1 ext4   /media/duchess/stuff
[...]
$ umount /media/duchess/stuff
```

Then use *parted* to delete the partition:

```
$ sudo parted /dev/sdb
GNU Parted 3.2
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) print
```

```
Model: ATA SAMSUNG HD204UI (scsi)
Disk /dev/sdb: 2000GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start   End     Size    File system  Name  Flags
 1      1049kB  1656GB  1656GB  ext4          stor-1
 1      1656GB  2656GB  1000GB  ext4          stor-2
(parted) rm 1
```

If you prefer a graphical tool, use GParted (Chapter 9).

Discussion

Yes, the command is *umount*, not *unmount*. *umount* dates from the ancient Unix era, when identifiers had a limit of six characters.

Deleting all the files in a partition does not delete the filesystem. The filesystem structure remains in place.

See Also

- *man 1 dd*

11.5 Using a New Filesystem

Problem

You just created a nice new filesystem, and you need to mount it.

Solution

After creating your new filesystem, you must create a mountpoint, and optionally configure automatic mounting. As discussed in the introduction to this chapter, a new filesystem must be mounted, or attached, to the running filesystem to be usable.

Ext4, XFS, and Btrfs all have access controls. If you want the files on these filesystems available to anyone other than the root user, you must adjust ownership and permissions. FAT16/32 and exFAT do not have access controls and are wide open to anyone.

Start by mounting your new filesystem. Create a mountpoint, which is a directory, and then mount the filesystem, like this example for Mad Max:

```
$ sudo mkdir -p /mnt/madmax/newfs
$ sudo mount /dev/sdb1 /mnt/madmax/newfs
```

The following example sets the ownership of the new filesystem to Mad Max, read-write-execute, with read-only permissions for group and world:

```
$ sudo chown -R madmax:madmax /mnt/madmax/newfs
$ sudo chmod -R 0755 /mnt/madmax/newfs
```

Now Mad Max can access the new filesystem. This mount only lasts until the next system restart; see Recipe 11.6 to learn how to configure automatic filesystem mounts.



Only One Filesystem per Mountpoint

Every filesystem needs its own unique mountpoint; you cannot put multiple filesystems on a single mountpoint.

Discussion

See Chapter 6 for detailed recipes on managing ownership and permissions.

The traditional directories that contain mountpoints are */mnt* and */media*. */mnt* is traditionally for static mounts (configured in */etc/fstab*), and */media* is for automounting removable media. You may create your mountpoints wherever you want. The advantage of using the traditional directories is having your mountpoints in a limited number of predictable locations.

A shared directory with mountpoints for multiple users could look like this, with a directory for each user:

```
$ tree /shared
/shared
├── duchess
├── madmax
└── stash
```

Then every filesystem needs its own mountpoint in the user subdirectories. For example, Mad Max has two filesystems mounted at *madmax1* and *madmax2*:

```
$ tree -L 2 /mnt
/mnt
├── duchess
├── madmax
│   ├── madmax1
│   └── madmax2
└── stash
```

The mountpoints can have any names you want. For example, Mad Max's mountpoints could be *fs1* and *fs2*, or *fred* and *ethel*, or *max1* and *max2*, whatever helps you remember what they are.

Use the *stat* command to see the permissions on a filesystem, like this example for Mad Max's new filesystem:

```
$ stat /shared/madmax/madmax1
[...]
Access: (0755/drwxr-xr-x) Uid: ( 0/ madmax) Gid: ( 0/ madmax)
```

List all filesystem mounts with *mount*:

```
$ mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs
[...]
```

Use the *mountpoint* command to learn if a directory is a mountpoint:

```
$ mountpoint madmax1/
madmax1/ is a mountpoint
```

See Also

- *man 1 chown*
- *man 1 chmod*
- *man 1 stat*

11.6 Creating Automatic Filesystem Mounts

Problem

You have added a new filesystem, and you want it to automatically mount at system startup.

Solution

This is what your */etc/fstab* file is for. The following example is added to the existing */etc/fstab* file to create a static mount for the filesystem in Recipe 11.5, and it will be automatically mounted at startup:

```
#<file system>    <mount point>        <type>   <options>       <dump>  <pass>
LABEL=xfs-ehd     /mnt/madmax/newfs    xfs      defaults,user   0       2
```

Use the *findmnt* command to test your new configuration:

```
$ sudo findmnt --verbose --verify
/
   [ ] target exists
   [ ] UUID=102a6fce-8985-4896-a5f9-e5980cb21fdb translated to /dev/sda2
   [ ] source /dev/sda2 exists
```

```
   [ ] FS type is btrfs
   [W] recommended root FS passno is 1 (current is 0)
/mnt/madmax/newfs
   [ ] target exists
   [ ] LABEL=xfs-ehd translated to /dev/sdb1
   [ ] source /dev/sdb1 exists
   [ ] FS type is xfs
[...]
0 parse errors, 0 errors, 1 warning
```

The warning "recommended root FS passno is 1 (current is 0)" is not significant. If that is the only warning, and there are no errors, reboot to test, or run the following command to mount your new */etc/fstab* entry:

```
$ sudo mount -a
```

Discussion

This is what the six *fstab* columns are for:

device
: The UUID or filesystem LABEL. Don't use */dev* names because they are not unique, and sometimes they change. Run *lsblk -o UUID,LABEL* to list UUIDs and filesystem labels to use in the *device:* column.

mountpoint
: The directory you created for the filesystem.

type
: The filesystem type, for example, *xfs*, *ext4*, or *btrfs*. You may use *auto* for the filesystem type, and the kernel will automatically detect the filesystem type.

options
: Your mount options in a comma-delimited list (see below for a list).

dump
: If you're using the *dump* command for backups, this tells *dump* the backup interval, in days. So, 1 means every day, 2 means every other day, 3 is every third day, and so on. Most likely you are not using *dump* and should enter 0.

pass
: This tells the filesystem checker which filesystem to check first at bootup, if it ever needs to. Make your root filesystem 1, any other Linux filesystems 2, and non-Linux filesystems 0.

The following *options* define permissions:

defaults
: The default options are *rw*, *suid*, *dev*, *exec*, *auto*, *nouser*, and *async*. The *defaults* values are overridden by appending additional options, for example *defaults,user*

gives the user permission to mount and unmount the filesystem. You may append as many options as you like, or omit *defaults* and list only the options you want.

rw
: Read/write.

ro
: Read-only.

suid
: Allow setuid and setgid bits to operate.

dev
: Interpret block and character devices.

exec
: Allow binaries to run.

auto
: Indicates which filesystems should start at boot.

nouser
: Nonroot users cannot mount or unmount the filesystem.

async
: Asynchronous I/O, which is standard for Linux.

user
: Nonroot users can mount and unmount the device, if they mounted it.

users
: Any user can mount and unmount the device.

noauto
: Do not automatically mount at boot.

ro
: Mount the filesystem read-only.

noatime
: Do not update the "time accessed" file attribute. *noatime* was used in times past to speed up performance. If you are on a modern computer, it probably won't make much difference.

gid
: Limit access to a group (from */etc/group*); for example, *gid=group1*.

See Also

- *man 8 mount*
- *man 5 fstab*
- systemd

11.7 Creating Ext4 Filesystems

Problem

You want to create a new Ext4 filesystem on an internal or external storage disk.

Solution

Start with a partition of the size you want for your filesystem. Then use the *mkfs.ext4* command to create the new Ext4 filesystem.

The following example overwrites an existing XFS filesystem with a new Ext4 filesystem. When you overwrite an existing filesystem, it must first be unmounted. In this example, the filesystem on */dev/sdb1* is mounted at */media/duchess/stuff*, which you can see with the *df* command:

```
$ df -Th /media/duchess/stuff/
Filesystem     Type  Size  Used Avail Use% Mounted on
/dev/sdb1      xfs   952M  7.9M  944M   1% /media/duchess/stuff
```

You may need root permissions to unmount:

```
$ sudo umount /media/duchess/stuff
```

Create the new Ext4 filesystem:

```
$ sudo mkfs.ext4 -L 'mylabel' /dev/sdb1
mke2fs 1.44.1 (24-Mar-2018)
/dev/sdb1 contains a XFS file system labelled 'stuff'
        created on Sun Sep 20 19:37:43 2020
Proceed anyway? (y,N) y
Creating filesystem with 466432 4k blocks and 116640 inodes
Filesystem UUID: 99da2e5d-f96a-4fb6-990d-599cf56247a2
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912

Allocating group tables: done
Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done
```

You could also create a new partition and put your new filesystem on it; see the examples for creating new partitions in Recipes 8.4 and 9.4.

Discussion

Overwriting a filesystem destroys all the data on it.

The *-L* option is for creating a volume label. This can be anything you want, up to 16 characters (FAT32 is limited to 11 characters). It is not required, though filesystem labels are useful, and for some operations, such as in */etc/fstab*, can be used in place of the long UUID.

The *-n* option does a dry run, so you see what will happen without actually creating the new filesystem.

mke2fs has numerous options, but you will likely use just a few of them: device name, volume label, dry-run, and creating an external journal. Its defaults are set in */etc/mke2fs.conf*, and I suggest not changing them without thorough study of the available settings.

See Also

- *man 8 mke2fs*
- Recipe 8.4
- Recipe 11.5

11.8 Configuring the Ext4 Journal Mode

Problem

You know that the default journal mode for ext4 is *data=ordered*, which does not journal data, but only metadata. It is a good balance of safety and speed, but you want to set it to *data=journal*, which is the safest.

Solution

Use the *tune2fs* command. First check your existing journal mode with *dmesg*. The filesystem must be mounted:

```
$ dmesg | grep sdb1
[25023.525279] EXT4-fs (sdb1): mounted filesystem with ordered data mode.
```

That confirms */dev/sdb1* is formatted as Ext4 and has the default *data=ordered* journal mode. Now change it to *data=journal* mode:

```
$ sudo tune2fs -o journal_data /dev/sdb1
tune2fs 1.44.1 (24-Mar-2018)
```

Unmount and remount, and check again with *dmesg*:

```
$ dmesg | grep sdb1
[25023.525279] EXT4-fs (sdb1): mounted filesystem with ordered data mode.
```

If you see multiple lines with conflicting information, like this:

```
[  206.076123] EXT4-fs (sdb1): mounted filesystem with journalled data mode.
[  206.076433] EXT4-fs (sdb1): mounted filesystem with ordered data mode.
```

Reboot, and then you should see only the "mounted filesystem with journalled data mode" line.

Discussion

The journal mode command options are named differently, depending on what documentation you are reading. In *man 8 tune2fs*, the following options are listed:

- journal_data
- journal_data_ordered
- journal_data_writeback

In the kernel documentation, and a whole lot of how-tos, these are the options:

- data=journal
- data=ordered
- data=writeback

The *data=* options are meant to be passed to the kernel at boot either in your bootloader configuration, or in */etc/fstab*. I favor using *tune2fs* because it is fast and easy, and works on all Ext4 filesystems regardless of their mount configurations.

These are the journal modes in order of data safety:

data=journal
: Provides the most protection for your data. All data and metadata are first written to the journal, and then written to the filesystem. In the event of a failure, this gives you the best chance of recovering your data. This is also the most resource intensive, as your changes are written twice.

data=ordered
: This does not write your data to the journal. Data is first written to the filesystem, and then metadata is written to the journal. The metadata is logically grouped in order and held in a single transaction. When the metadata is written to disk, its associated data blocks are written first.

data=writeback
: This is the fastest and the least safe. Data is first written to the filesystem, and then metadata is written to the journal. Data ordering is not preserved. I don't think the small performance gain is worth the extra risk.

See Also

- *man 8 tune2fs*
- Kernel documentation for the Ext4 filesystem

11.9 Finding Which Journal Your Ext4 Filesystem Is Attached To

Problem

You have several Ext4 filesystems, some with internal journals and some with external journals, and you want to know which journals they are using.

Solution

Meet a new command, *dumpe2fs*. This is a part of the *e2fsprogs* suite of ext2/3/4 utilities. Query your Ext4 filesystem:

```
$ sudo dumpe2fs -h /dev/sda1 | grep -i uuid
dumpe2fs 1.43.8 (1-Jan-2018)
Filesystem UUID:          8593f3b7-4b7b-4da7-bf4a-cc6b0551cff8
Journal UUID:             f8e42703-94eb-49af-a94c-966e5b40e756
```

The *Journal UUID* belongs to the journal. Run *lsblk* to verify details:

```
$ lsblk -f | grep  f8e42703-94eb-49af-a94c-966e5b40e756
└─sdb5 ext4    journal1 f8e42703-94eb-49af-a94c-966e5b40e756
```

And there it is. An Ext4 filesystem using an internal journal looks like this, without the Journal UUID line:

```
$ sudo dumpe2fs -h /dev/sda2 | grep UUID
dumpe2fs 1.44.1 (24-Mar-2018)
Filesystem UUID:          64bfb5a8-0ef6-418a-bb44-6c389514ecfc
```

Discussion

There is always a way to find out where things are in Linux. The *dumpe2fs* command shows a lot of useful information about your Ext4 filesystems, including UUIDs, filesystem creation time, block count, free blocks, journal size, and much more.

See Also

- *man 8 dumpe2fs*

11.10 Improving Performance with an External Journal for Ext4

Problem

You have heard that placing the Ext4 journal on a different disk than the filesystem improves performance, and you want to do this.

Solution

An external journal improves performance when your journal mode is *data=journal*. (See the Discussion for more information on journal modes.) You may create a new Ext4 filesystem and external journal, or convert an existing filesystem to use an external journal.

The two disks must be on the same machine and have similar read and write speeds. If the journal disk is slower than the filesystem disk, you will not see much, if any, of a performance gain. You could use two similar solid-state disks (SSDs), two similar hard disk drives (HDDs), or use a small SSD for the journal and a large HDD for the filesystem, because SSDs are much faster than HDDs.

Locating the Ext4 journal on a separate disk takes several steps. In the following example, we will create two new partitions, one for the journal and one for the new Ext4 filesystem. Then create the journal, then the filesystem, and attach it to the journal.

The first partition is for the journal on */dev/sdb5*, 200 GB in size, and the second partition is for the Ext4 filesystem on */dev/sda1*, 500 GB:

```
$ sudo parted
(parted) select /dev/sdb
Using /dev/sdb
(parted) mkpart "journal1" ext4 1600GB 1800GB
(parted) select /dev/sda
```

```
Using /dev/sda
(parted) mkpart "ext4fs" ext4 1MB 500GB
```

The external journal and the filesystem must have the same block size, which is specified in the following example with *-b 4096*. If you don't know the block size, find it with *tune2fs*. The following commands are run in the Bash shell, and not in the *parted* shell:

```
$ sudo tune2fs -l /dev/sda1  | grep -i 'block size'
Block size:               4096
```

Now create the journal, which can take a few minutes, and then the new filesystem:

```
$ sudo mke2fs -b 4096 -O journal_dev /dev/sdb5
mke2fs 1.43.8 (1-Jan-2018)
/dev/sdb2 contains a ext4 file system labelled 'ext4'
        created on Mon Jan  4 18:25:30 2021
Proceed anyway? (y,N) y
Creating filesystem with 48747520 4k blocks and 0 inodes
Filesystem UUID: f8e42703-94eb-49af-a94c-966e5b40e756
Superblock backups stored on blocks:
Zeroing journal device:

$ sudo mkfs.ext4 -b 4096 -J device=/dev/sdb5 /dev/sda1
mke2fs 1.43.8 (1-Jan-2018)
Creating filesystem with 35253504 4k blocks and 8814592 inodes
Filesystem UUID: 8593f3b7-4b7b-4da7-bf4a-cc6b0551cff8
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
        4096000, 7962624, 11239424, 20480000, 23887872

Allocating group tables: done
Writing inode tables: done
Adding journal to device /dev/sdb2: done
Writing superblocks and filesystem accounting information: done
```

You're finished and can use your new filesystem.

You can attach an external journal to an existing filesystem with the *tune2fs* command. First clear the journal on the existing filesystem, then link the filesystem to the external journal:

```
$ sudo tune2fs -O ^has_journal /dev/sda1
$ sudo tune2fs -b 4096 -J device=/dev/sdb5 /dev/sda1
```

Discussion

The Ext4 journal provides extra protection for your data, in the event of a disk or system failure, by tracking changes that are not yet written to disk. Even if it loses your most recent changes, it protects the filesystem from being corrupted, so you lose just a little bit instead of the whole works.

Moving the journal to a separate disk on the same machine provides a noticeable performance boost when the journal mode is *data=journal*. Ext4 has three journaling modes: *journal*, *ordered*, and *writeback*. The default is *ordered*. See Recipe 11.8 to learn about these modes and how to select the one you want to use.

The caret, ^, disables a feature. In the example in the recipe, it clears the existing internal journal.

Ext4 journals cannot be shared, and can be used by only one filesystem.

See Also

- *man 8 mke2fs*
- *man 8 tune2fs*
- Chapter 8
- Chapter 9

11.11 Freeing Space from Reserved Blocks on Ext4 Filesystems

Problem

Most Linux distributions reserve 5% of Ext4 filesystems for the root user and system services. On large modern hard disks that is a lot of space, and you want to free some of that space.

Solution

Use the *tune2fs* command to adjust the size of the free space on an Ext4 filesystem. You may configure it by percentage, like this example that reduces it to 1%:

```
$ sudo tune2fs -m 1 /dev/sda1
tune2fs 1.44.1 (24-Mar-2018)
Setting reserved blocks percentage to 1% (820474 blocks)
```

That is still about 3 gigabytes, with 4K blocks (820,474 x 4,096 = 3,360,661,504 bytes). Find your block size:

```
$ sudo tune2fs -l /dev/sda1  | grep -i 'block size'
Block size:               4096
```

You can set a fractional percentage:

```
$ sudo tune2fs -m .25 /dev/sda1
tune2fs 1.44.1 (24-Mar-2018)
Setting reserved blocks percentage to 0.25% (205118 blocks)
```

That is roughly 800 MB. Or, specify a number of blocks:

```
$ sudo tune2fs -r 250000 /dev/sda1
tune2fs 1.44.1 (24-Mar-2018)
Setting reserved blocks count to 250000
```

250,000 4K blocks is about a gigabyte. Check your work:

```
$ sudo tune2fs -l /dev/sda1 | grep -i 'reserved block'
Reserved block count:     250000
```

Discussion

If you run out of disk space, you can still log in as root and free up space, which you could not do if that 5% was not held in reserve. However, that 5% is a holdover from the days of megabyte hard disks. Hard disks are so large now, you don't need all that reserved space. For example, 5% of a 1 TB disk is about 50 GB. Only a few hundred megabytes of reserved space is necessary. I set mine to a gigabyte. It's easy to remember and provides more than enough room.

Use the *dumpe2fs* command to check out the reserved blocks settings in your Ext4 filesystem:

```
$ sudo dumpe2fs -h /dev/sda1
[...]
Block count:              82047488
Reserved block count:     250000
[...]
```

See Also

- *man 8 dumpe2fs*
- *man 8 tune2fs*

11.12 Creating a New XFS Filesystem

Problem

You like XFS, and want to create a new XFS filesystem.

Solution

You need the *xfsprogs* package installed on your system and a partition for the new filesystem. Then create your new XFS filesystem with *mkfs.xfs*. The following example, on Ubuntu, demonstrates all these steps. The example new partition is */dev/sda1*, and the new filesystem gets an *xfstest* label:

```
$ sudo apt install xfsprogs
$ sudo parted /dev/sda mkpart testxfs xfs 1MB 500GB
$ sudo mkfs.xfs -L xfstest /dev/sda1
meta-data=/dev/sdb5              isize=512    agcount=4, agsize=640000 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=1, sparse=0, rmapbt=0,
reflink=0
data     =                       bsize=4096   blocks=2560000, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0 ftype=1
log      =internal log           bsize=4096   blocks=2560, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

Check your work with *lsblk*:

```
$ lsblk -f | grep -w sda1
├─sda1 xfs    xfstest  bb5dddb3-af74-4bed-9d2a-e79589278e84
```

Mount your new filesystem, adjust ownership and permissions, and it's ready to use. The following example mounts it on */mnt/xfstest*, sets ownership to Duchess, read-write for Duchess and read only for everyone else:

```
$ sudo mkdir /mnt/xfstest
$ sudo mount /dev/sda1 /mnt/xfstest
$ sudo chown -R duchess:duchess /mnt/xfstest
$ sudo chmod -R -755 /mnt/xfstest
```

Discussion

The command output from creating a new XFS filesystem contains a few helpful items, like the block size, number of blocks, and sector size.

See Also

- *man 8 mkfs.xfs*

11.13 Resizing an XFS Filesystem

Problem

You want to resize an XFS filesystem.

Solution

You can only increase the size of an XFS filesystem. If you need it to be smaller, you must copy your data to a safe location, create a smaller partition, format it as XFS, then restore your data.

Increasing the size is less work. You need free space at the end of the partition that your XFS filesystem is on. In the following examples, the new endpoint for the partition is 2700 GB, and the filesystem is mounted at */media/duchess/xfs*.

Launch *parted*. Print the partition information to verify the correct partition and endpoint, increase the partition size, then quit *parted*:

```
$ sudo parted /dev/sdb
GNU Parted 3.3
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) p free
Model: ATA SAMSUNG HD204UI (scsi)
Disk /dev/sdb: 4000GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start   End     Size    File system  Name   Flags
        17.4kB  1049kB  1031kB  Free Space
 1      1049kB  1656GB  1656GB  xfs          files
 2      1656GB  1759GB  103GB   xfs          files2
        1759GB  4000GB  242GB   Free Space

(parted) resizepart 2
(parted) Warning: Partition /dev/sdb2 is being used. Are you sure you want to
continue?
Yes/No? Yes
End?  [1759GB]? 1900GB
(parted) q
```

Now, expand the filesystem to match the new partition size:

```
$ sudo xfs_growfs /media/duchess/xfs
```

You're done! Enjoy your new larger filesystem.

Discussion

You also have the option to unmount the filesystem and resize it offline. This is a little safer.

Using GParted to resize a filesystem is fast and easy; see Recipe 9.7.

See Also

- Recipe 8.8
- Recipe 9.7

11.14 Creating an exFAT Filesystem

Problem

Your digital camera flash drive is formatted with the exFAT filesystem, or you have other flash storage devices that use exFAT, and you want to read, write, and edit the files from these devices on your Linux system.

Solution

There are two possible solutions: one is to use the exFAT implementation that runs on Filesystem in Userspace (FUSE). The other solution is to use the native implementation that runs in the Linux kernel, rather than userspace. In this recipe we will use exFAT FUSE because at the time this was written the native implementation had not yet made it into most distribution releases. Look for kernel version 5.7, and check your distribution release notes and news. (Run the *uname -r* command to see your kernel version.)

The exFAT package names vary. *exfat-fuse* and *exfat-utils* are the older packages. *exfatprogs* is the newest implementation, replacing both *exfat-fuse* and *exfat-utils*. Whatever you have, go ahead and install it.

The command to create a new exFAT filesystem is the same for both. The following example formats */dev/sdc1* as exFAT:

```
$ sudo mkfs.exfat /dev/sdc1
mkexfatfs 1.2.8
Creating... done.
Flushing... done.
File system created successfully.
```

exFAT is designed to be simple, so there are not a lot of options. You can give it a label:

```
$ sudo exfatlabel /dev/sdc2 exfatfs
```

Verify your changes with *lsblk*:

```
$ lsblk -f
NAME   FSTYPE LABEL   UUID
sdc
├─sdc1
├─sdc2 exfat  exfatfs 8178-51D4
└─sdc3
```

Discussion

You do not need a special exFAT partition to read exFAT files on other devices, but only exFAT installed on your Linux system.

If you prefer a graphical partitioning tool, GParted does not support exFAT, due to legal concerns. GNOME Disks, called Disks in most GNOME implementations, does support exFAT. You do not have to install GNOME to get Disks; look for the *gnome-disk-utility* package.

Microsoft released the exFAT specification in 2019. Samsung wrote *exfatprogs* and released it in early 2020. By the time you read this, the latest releases of Fedora, Ubuntu, and openSUSE Tumbleweed should have native exFAT support.

See Also

- *man 8 exfat*
- *man 8 exfatlabel*

11.15 Creating FAT16 and FAT32 Filesystems

Problem

You need to know how to create FAT16 and FAT32 filesystems.

Solution

You need the *dosfstools* package, which is installed by default on most Linuxes. The following examples demonstrate creating a new 500 MB partition with *parted*, then formatting the partition with FAT32.

Create the new partition, and note how to change the measurement units to MB, and how to use *mkpart* interactively:

```
$ sudo parted /dev/sdb
GNU Parted 3.2
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) print
Model: ATA SAMSUNG HD204UI (scsi)
Disk /dev/sdb: 2000399MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start    End     Size    File system  Name   Flags
1       0.00GB   1656GB  1656GB  xfs          files
```

```
(parted) unit mb
mkpart
Partition name?  []?
File system type?  [ext2]? fat32
Start? 1656331MB
End? 1656831MB
(parted) print
Model: ATA SAMSUNG HD204UI (scsi)
Disk /dev/sdb: 2000399MB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start      End        Size       File system  Name  Flags
 1      1.05MB     1656331MB  1656330MB  xfs          bup
 2      1656331MB  1656831MB  500MB      fat32

(parted) q
```

The *partition* name is optional; in the example it is left empty. Now create a nice new FAT32 filesystem:

```
$ sudo mkfs.fat -F 32 -n fat32test /dev/sdb2
mkfs.fat 4.1 (2017-01-24)
mkfs.fat: warning - lowercase labels might not work properly with DOS or Windows
```

Verify with *lsblk*:

```
$ lsblk  -f /dev/sdb
NAME   FSTYPE LABEL       UUID                                 FSAVAIL FSUSE% MOUNTPOINT
sdb
├─sdb1 xfs    xfstest     1d742b2d-a621-4454-b4d3-469216a6f01e
└─sdb2 vfat   fat32test   AB39-1808
```

Discussion

If you want a FAT16 filesystem, use *-F 16*.

FAT16 files and filesystems max out at 4 GB.

FAT32 supports a maximum file size of 4 GB, and a maximum partition size of 16 TB, using 4 KB sectors and 64 KB clusters.

See Also

- Chapter 8
- Chapter 9
- *man 8 mkfs.fat*

11.16 Creating a Btrfs Filesystem

Problem

Btrfs sounds cool, and you want to try it out.

Solution

It is cool, and it is also complex. SUSE Linux Enterprise Server (SLES) and openSUSE are the best Linux distributions to try Btrfs on. SLES and openSUSE are the biggest Btrfs supporters and developers, and they created the excellent Snapper tool for managing Btrfs snapshots. They also provide the most thorough documentation. The default partitioning on an openSUSE/SLES sets up Btrfs subvolumes and automatic snapshots.

Start by downloading the latest openSUSE Tumbleweed. Launch the installer, and when you get to the Suggested Partitioning screen, take a look at the installer's first proposal (Figure 11-2).

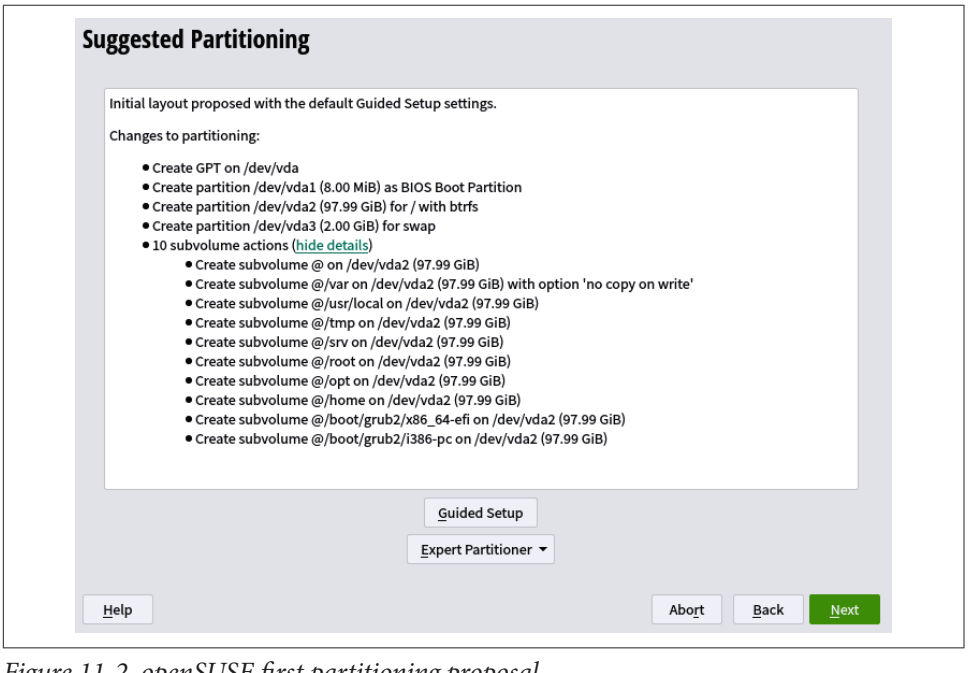


Figure 11-2. openSUSE first partitioning proposal

Click Guided Setup to modify this proposal. Skip past the "Enable logical volume management (LVM) / Enable disk encryption" screen, and stop at the Filesystem Options screen. Select "Propose Separate Home Partition," and format it as Btrfs.

Check both boxes for "Propose Separate Swap Partition," then click Next (Figure 11-3).

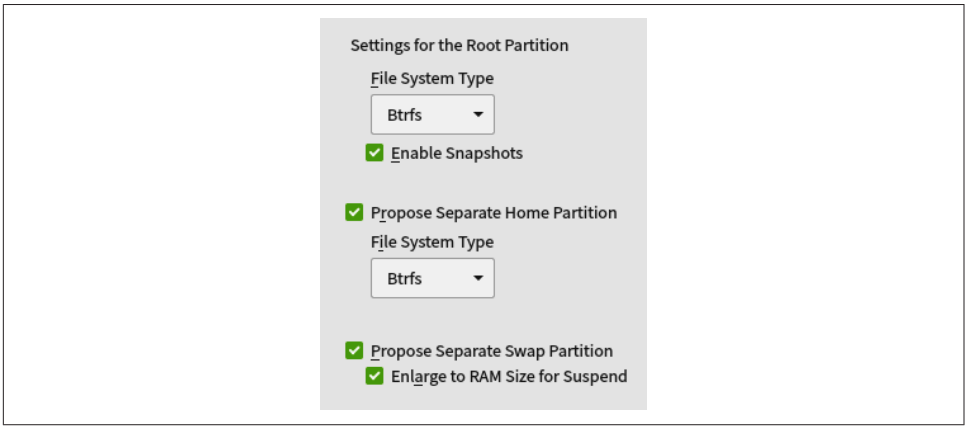


Figure 11-3. Create a home partition

This returns you to the Suggested Partitioning screen. If you wish to adjust partition sizes, click Expert Partitioner → Start with Current Proposal (Figure 11-4). Otherwise click Next and continue with the installation.

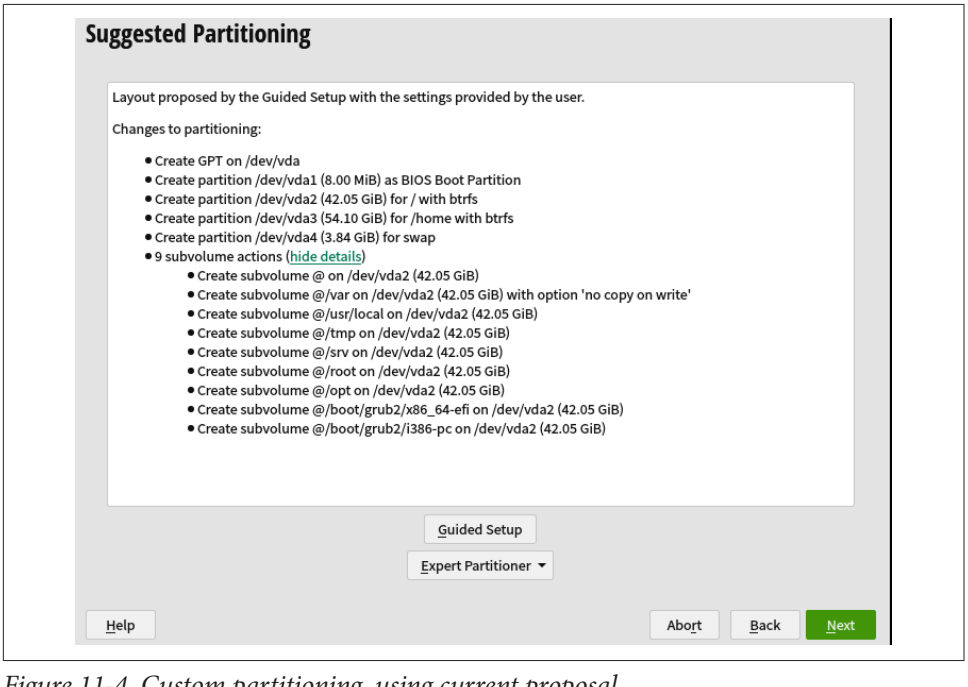


Figure 11-4. Custom partitioning, using current proposal

When you are finished, you will have a ready-to-use Btrfs Linux system, already set up with good defaults.

Discussion

Setting up Btrfs manually is a bit of a chore, though as you learn about it you might want to try setting it up manually. I like to learn new things by starting with a working implementation. It is not possible, at least for me, to provide a useful Btrfs how-to in a few recipes. Btrfs is so flexible and so capable, it needs its own book. Which it has, thanks to the good SUSE people. Consult the Startup Guide for installation, and the "System Recovery and Snapshot Management with Snapper" section in the openSUSE documentation. Snapper + Btrfs is a great combination for Btrfs management and fast failure recovery.

See Also

- The openSUSE Startup and Reference Guides
- The Deployment and Administration guides in the SLES Product Manuals

CHAPTER 12

Secure Remote Access with OpenSSH

OpenSSH is the tool of choice for secure remote administration. It encrypts authentication and all traffic during a session, and guarantees the integrity of the data transfer. If something happens to alter your packets, SSH will tell you. In this chapter you will learn how to set up SSH access to remote hosts, manage your SSH encryption keys, configure logins to multiple remote hosts, customize your Bash prompt to show when it is an SSH session, and more good things.

OpenSSH supports a large number of strong encryption algorithms. All of them are unencumbered by patents because the OpenSSH team has gone to great lengths to ensure that no patented or otherwise encumbered code is inside OpenSSH. Recipe 12.16 shows how to print lists of all supported algorithms.

OpenSSH is a suite of remote transfer utilities:

- *sshd*, the OpenSSH server daemon.
- *ssh*, short for secure shell, though it doesn't really include a shell, but provides a secure channel to the command shell on the remote system.
- *scp*, secure copy, for encrypted file transfer.
- *sftp*, Secure File Transfer Protocol, provides file access.
- *ssh-copy-id*, a nice little program for installing your public key to a remote SSH server's *authorized_keys* file.
- *ssh-keyscan*, finds and collects public host keys on a network, saving you the trouble of hunting them down manually.
- *ssh-keygen*, generates and manages authentication keys.
- *ssh-add*, adds your identities to the authentication agent, *ssh-agent*.

In this chapter you will learn about *ssh*, *sshd*, *ssh-copy-id*, *ssh-keygen*, and two useful related utilities: *sshfs* and *ssh-agent*.

sshfs mounts remote filesystems on your local PC, while *ssh-agent* remembers the passphrases on your private SSH keys over multiple SSH logins for automatic authentication. *ssh-agent* binds to a single login session, so logging out or opening another terminal means starting over. A better utility for automated operations is Keychain, which is a frontend to *ssh-agent*. Keychain reuses *ssh-agent* until you restart your machine, so you only have to enter your passphrases at startup (see Recipe 12.10).

OpenSSH supports different types of authentication:

Password authentication
: Uses your Linux login and password to authenticate. This is the simplest and the most flexible, because you can log in from any machine. You must be careful to not open an SSH session from an untrustworthy computer, like in a library or internet cafe. If it is infected with a keylogger, it will capture your credentials.

Public key authentication
: Authenticates with your personal SSH public keys, not your system login. This is a bit more work to set up because you need to create and distribute your public keys, and you can log in only from machines that hold your private key. Some commercial services require customers to use some form of public key authentication.

Passphrase-less authentication
: Public key authentication without a passphrase. This is useful for automated services, like scripts and cron jobs. Anyone who succeeds in thieving the private key can easily masquerade as you, so you need to be very protective of a passphrase-less private key.

An alternative to using keys without passphrases is Keychain, which remembers your private keys for you (see Recipe 12.10).

There are two different uses for authentication keys: host keys, which authenticate computers, and public keys, which authenticate users. SSH keys come in pairs, private and public. Transmissions are encrypted with the public key and decrypted with the private key, a brilliantly simple scheme. You can safely distribute your public keys as much as you want, while you must protect your private key and not let anyone else have it.

Server and client are defined by the direction of the transaction. The server has the SSH daemon running and listening for connection requests, and the client is anyone logging in to this machine via SSH.

12.1 Installing OpenSSH Server

Problem

You want to install an OpenSSH server.

Solution

Most Linux distributions install the OpenSSH client by default, but not always the server. The different Linux distributions package OpenSSH in different ways, so use your package manager to list the packages for your Linux (see the Appendix). Install the server, then check if it has started:

```
$ systemctl status sshd
● sshd.service - OpenSSH Daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; disabled; vendor preset
   Active: inactive (dead)
   [...]
```

This shows that the server is not running and is not enabled. On most Linuxes, OpenSSH is not configured to start automatically after installation. This is good because you need to configure your server correctly before opening it up to receive connection requests. If it is running before you have examined the server configuration, stop it, or block its listening port(s) with your firewall.

The next steps are to set up host encryption keys and configure your server. See Recipes 12.2 and 12.3.

Discussion

Remember, server and client are not only about hardware, but are defined by the direction of the transaction. The server has the SSH daemon running and listening for connection requests, and the client is anyone logging in to the server via SSH. Any Linux PC can be a server, client, or both.

See Also

- Chapter 14
- OpenSSH
- sshd (8)
- The Appendix

12.2 Generating New Host Keys

Problem

Your Linux distribution does not automatically create host keys at installation, or you want to replace your existing host keys, or when you clone an installation or a virtual machine your clones need their own unique host keys.

Solution

Use the *ssh-keygen* command. There are four different types of keys: RSA, DSA, ECDSA, and ED25519. First, delete the old keys, if they exist:

```
$ sudo rm /etc/ssh/ssh_host*
```

Create all of the new keys at once with the following command:

```
$ sudo ssh-keygen -A
ssh-keygen: generating new host keys: RSA DSA ECDSA ED25519
```

Discussion

If you ever get bored and need something to do, try researching "Which SSH key formats should I use?" The arguments are endless. The short answer is use RSA, ECDSA, and ED25519, and avoid DSA. Delete your DSA host key and keep the rest.

RSA is the oldest. It is strong and provides the most compatibility.

ECDSA and ED25519 are newer, very strong, and computationally less expensive.

Some older SSH clients do not support ECDSA and ED25519. Hopefully you are not using such ancient clients, because ECDSA and ED25519 were released with OpenSSH 6.5 in 2014. It is extremely important to keep security services updated and to not allow unsafe old clients.

See Also

- OpenSSH
- ssh-keygen (1)

12.3 Configuring Your OpenSSH Server

Problem

You want to configure your OpenSSH server as securely as possible and test it safely.

Solution

First, verify that your server's private host keys are owned by root, read-only:

```
$ ls -l /etc/ssh/
-r-------- 1 root root    227 Jun  4 11:30 ssh_host_ecdsa_key
-r-------- 1 root root    399 Jun  4 11:30 ssh_host_ed25519_key
-r-------- 1 root root   1679 Jun  4 11:30 ssh_host_rsa_key
```

That is how they are supposed to look. Then check your public keys, which are owned by root, read-write for root, and read-only for everyone else:

```
$ ls -l /etc/ssh/
-rw-r--r-- 1 root root    174 Jun  4 11:30 ssh_host_ecdsa_key.pub
-rw-r--r-- 1 root root     94 Jun  4 11:30 ssh_host_ed25519_key.pub
-rw-r--r-- 1 root root    394 Jun  4 11:30 ssh_host_rsa_key.pub
```

These are correct.

Now take a look at */etc/ssh/sshd_config*. When you change this file, reload *sshd* to load your changes:

```
$ sudo systemctl reload sshd.server
```

Uncomment the options you want to use or change.

Configure *sshd* to check if the file modes and ownership of the user's files and home directory are correct before accepting their login:

```
StrictModes yes
```

If file permissions are not correct, this setting will not allow them to log in.

If your machine has more than one IP address, define which address, or addresses, it listens on:

```
ListenAddress 192.168.10.15
ListenAddress 192.168.10.16
```

You may assign nonstandard ports for *sshd* to listen on. Use only ports above 1024, and check */etc/services* to find unused ports, then add your new ports to */etc/services*:

```
sshd 2022
sshd 2023
```

Then add them to */etc/ssh/sshd_config*:

```
Port 2022
Port 2023
```

You can restrict access to only the specified groups (create these groups in */etc/group*):

```
AllowGroups webadmins backupadmins
```

Or deny access with *DenyGroups*.

Do not allow root logins. It is safer to log in as an unprivileged user, and then use *sudo* after login:

```
PermitRootLogin no
```

An alternative is to allow root logins only with public key authentication:

```
PermitRootLogin prohibit-password
```

You can disable password logins for all users, and allow only public key authentication (see Recipe 12.7):

```
PasswordAuthentication no
```

You can deny specified users, either by username, or user at hostname or IP address:

```
DenyUsers duchess madmax stash@example.com cagney@192.168.10.25
```

Or allow access with *AllowUsers*. You may use both, and *DenyUsers* is always processed first.

Limit the length of time the server waits for a user to log in and complete the connection. The default is 120 seconds:

```
LoginGraceTime 90
```

You can limit the number of failed connection attempts. The default is 6:

```
MaxAuthTries 4
```

Discussion

Any port scanner will find your open ports, and attackers will attempt brute force password cracking. Attackers still target the default SSH port 22 the most. Changing the port won't reduce this risk very much, but it should reduce the number of entries in your log files. When you use alternate port numbers, first look in */etc/services* to find unused ports, and then record the ports you use in this file.

Public key authentication is very strong and cannot be brute-forced like password logins (see Recipe 12.7). The trade-off is less convenience, as you can log in only from machines that have your private key.

See Also

- OpenSSH
- *man 5 sshd_config*
- Recipe 12.5
- Recipe 12.7

12.4 Checking Configuration Syntax

Problem

Everyone makes mistakes, and you want a syntax checker for */etc/ssh/sshd_config*.

Solution

And you shall have one. After making your changes, run this command:

```
$ sudo sshd -t
```

If there are no syntax errors, it exits silently. If it find mistakes, it tells you:

```
$ sudo sshd -t
/etc/ssh/sshd_config: line 9: Bad configuration option: Porotocol
/etc/ssh/sshd_config: terminating, 1 bad configuration options
```

You can do this while the SSH daemon is running, so you can correct your mistakes before issuing a reload or restart command.

Discussion

The *-t* stands for *test*. It does not affect the SSH daemon, it only checks */etc/ssh/sshd_config* for syntax errors, so you can use it anytime.

See Also

- *man 5 sshd_config*
- OpenSSH

12.5 Setting Up Password Authentication

Problem

You want to set up your OpenSSH client to log in to a remote host using the simplest method that it supports.

Solution

Password authentication is the simplest way to set up remote SSH access. You need:

- OpenSSH server installed and properly configured on the machine you want to log in to (Recipe 12.3)
- The SSH daemon running on the remote machine, and port 22, or whatever port *sshd* uses, not blocked by firewalls

- The SSH client installed on your client machine
- Your own user account on the remote machine
- Host keys on the server (see Recipe 12.2)

The public host key must be distributed to the clients. The easy way is to log in from the client, and let OpenSSH transfer the key:

```
duchess@pc:~$ ssh duchess@server1
  The authenticity of host 'server1 (192.168.43.74)' can't be established.
  ECDSA key fingerprint is SHA256:8iIg9wwFIzLgwiiQ62WNLF5oOS3SL/aTw6gFrtVJTx8.
  Are you sure you want to continue connecting (yes/no)? *yes*
  Warning: Permanently added 'server1,192.168.43.74' (ECDSA) to the list of
  known hosts.
  Password: password
  Last login: Wed Jul  8 19:22:39 2021 from 192.168.43.183
  Have a lot of fun...
```

Now Duchess can work on *server1* just as if she were sitting at *server1*'s keyboard. All traffic and authentication are encrypted.

The host key exchange happens only once, the first time you log in. You should never be asked again unless the key is replaced with a new key, or you delete it from your personal *~/.ssh/known_hosts* file.

Discussion

server1's public host key is stored in the *~/.ssh/known_hosts* file on the client PC. This file can contain any number of host keys.

It is unsafe to log in as root over SSH; it is better to log in as an ordinary user, then *su* or *sudo* after login. You can log in as any user that has an account on the remote machine, if you know their password:

```
duchess@pc:~$ ssh madmax@server1
```

When you have the same username on both machines, you don't need to specify the user, and can log in like this:

```
duchess@pc:~$ ssh server1
```

I make it a habit to always specify the username as cheap insurance against mistakes.

Don't get too worked up over *client* and *server*. These are not about hardware. The server is whatever machine you are logging in to, and the client is wherever you are logging in from. *sshd* does not need to be running on the client.

There is a risk that the host key transmission could be intercepted and a forged key substituted, which would allow an attacker access to your systems. You can verify the public key fingerprint before typing **yes**. Use an old-fashioned method like writing it down and comparing, or a newfangled method like taking a photo of the host key

with your phone for comparison, or using your phone as an actual phone and calling someone who has access to the remote machine to read the fingerprint to you.

See Recipe 12.6 to learn how to retrieve a key fingerprint.

See Also

- Recipe 12.6
- OpenSSH
- *man 1 ssh*
- *man 1 ssh-keygen*
- *man 8 sshd*

12.6 Retrieving a Key Fingerprint

Problem

You need the fingerprint of a host key so you can verify for the client that the key is legitimate.

Solution

Use the *ssh-keygen* command on the server with the host key you want to query:

```
duchess@server1:~$ ssh-keygen -lf /etc/ssh/ssh_host_rsa_key
4096 SHA256:32Pja4+F2+MTdla9cs4ucecThswRQp6a4xZ+5sC+Bf0 backup server1 (RSA)
```

Discussion

This is where old-fashioned methods of communication, like telephone and sneakernet, come in handy. Don't use email, unless you already have encrypted email with its own separate encryption and authentication, because unencrypted email is easy to intercept and read.

See Also

- OpenSSH
- *man 1 ssh-keygen*

12.7 Using Public Key Authentication

Problem

You want to use public key authentication because it is stronger than password authentication, and because it does not use your Linux password. You want the option of using a single public key to access multiple systems, or creating a unique public key for each remote machine.

Solution

Yes, Linux user, you can have it all. You may create as many SSH keys as you want and use them however you wish. This is my favorite incantation for creating a new RSA key pair. Of course you will create your own comment and key name. (See the Discussion to learn if you need to set a passphrase on your private key.)

```
duchess@pc:~/.ssh $ ssh-keygen -C "backup server2" -f id-server2 -t rsa -b 4096
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in id-server2.
Your public key has been saved in id-server2.pub.
The key fingerprint is:
SHA256:32Pja4+F2+MTdla9cs4ucecThswRQp6a4xZ+5sC+Bf0 backup server2
The key's randomart image is:
+---[RSA 4096]----+
|          ..     |
|          ....   |
|           o. . .|
|          +  .  o|
|        S* .o o o|
|        +.+..Bo*+|
|         *.+*EX=o|
|        o *o.Oo+.|
|         o.o=+*+.|
+----[SHA256]-----+
```

The next step is to copy your nice new key to a remote machine, which in this case is the local backup server *server1*. You must already have SSH access to the remote machine, for example, via host key authentication, then use the *ssh-copy-id* command to transfer your public key to the server:

```
duchess@pc:~/.ssh $ ssh-copy-id -i id-server1 duchess@server1
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: ".ssh/id-server1"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are
prompted now it is to install the new keys

Number of key(s) added: 1
```

```
Now try logging into the machine, with:   "ssh 'duchess@server1'"
and check to make sure that only the key(s) you wanted were added.
```

Try logging in:

```
duchess@pc:~/.ssh $ ssh -i id-server1 duchess@server1
Enter passphrase for key 'id-server1':
Last login: Sat Jul 11 11:09:53 2021 from 192.168.43.234
Have a lot of fun...
duchess@server1:~$
```

You may use this new key to access multiple remote hosts, or create a unique key for each remote host. Using the same key for multiple machines is easy to use, but a pain to change on multiple hosts. If a unique key is compromised or lost, you only need to replace it once.

Discussion

Always use a passphrase on SSH keys created for human users, because anyone who gains access to your private keys can masquerade as you if there is no passphrase.

ssh-copy-id is a lovely little utility that ensures your public keys are copied into the correct location, which is *~/.ssh/authorized_keys* on the remote host, in the correct format and with the correct permissions. It also ensures your private key will not be copied by mistake.

Options are as follows:

- *-C* is for adding a comment to your key, which can help you remember what the key is for.
- *-f* is the key name, which can be anything you want. Be mindful of your current working directory; if you are not in *~/.ssh*, include the path.
- *-t* is the key type: *rsa*, *ecdsa*, or *ed25519*.
- *-b* is the bit strength, and only *rsa* takes this option. The default is 2048, and 4096 is the maximum. More bits equals more processing overhead, but it is doubtful you would notice any difference using 4096 bits except on old feeble hardware or on very busy servers.
- *-i* tells your SSH client which key you want to use. When you have more than one key, you must use this. When you have multiple public keys, you may see a "Too many authentication failures" error message if you do not specify one key, because SSH tries all of them when one is not specified.

See Also

- OpenSSH
- *man 1 ssh*
- *man 1 ssh-keygen*

12.8 Managing Multiple Public Keys

Problem

You want to use different keys for different servers. How do you manage keys with different names?

Solution

When you create a new key pair, use the *-f* option of the *ssh-keygen* command to give keys unique names:

```
duchess@pc:~/.ssh $ ssh-keygen -t rsa -f id-server2
```

Then, use the *-i* option to specify the key you want to use when you log in to the remote host:

```
duchess@pc:~/.ssh $ ssh -i id-server2 duchess@server2
```

To manage multiple public keys more easily, create a new file, *~/.ssh.config*. This file configures the logins for your various remote hosts, so you log in with *ssh foo* instead of a long command string. The following example configures a simpler login for Duchess to access *server2*:

```
Host server2
  HostName server2
  User duchess
    IdentityFile ~/.ssh/id-server2
    IdentitiesOnly yes
```

Now Duchess logs in like this, using the *Host* value:

```
$ ssh server2
```

Keep adding to this file for your other public key logins, like this:

```
Host server3
  HostName server3
  User duchess
    IdentityFile ~/.ssh/id-server3
    IdentitiesOnly yes

Host server3
  HostName server3
```

```
User madmax
  IdentityFile ~/.ssh/id-server3
  IdentitiesOnly yes
```

Discussion

In the preceding solution snippet:

- The *Host* line defines the start of each configuration. This is the label you use to login, and it can be anything you want.
- *HostName* is the remote machine's hostname, fully qualified domain name, or IP address.
- *User* is your user on the remote machine.
- *IdentityFile* is the full path to your public key.
- *IdentitiesOnly yes* tells *ssh* to use the settings in *~/.ssh/config*, or passed on the command line, and not other providers, if there are any.

The default SSH port number is 22. When you need to connect to a nonstandard port, for example 2022, specify it with *Port*:

```
Port 2022
```

You may call your keys anything you want. I like to use descriptive names so I know what machines they belong to.

Remember to always put a passphrase on your personal private keys.

See Also

- OpenSSH
- *man 1 ssh_config*
- *man 1 ssh*

12.9 Changing a Passphrase

Problem

You want to change the passphrase on one of your private keys.

Solution

Use the *-p* option with the *ssh-keygen* command:

```
$ ssh-keygen -p -f ~/.ssh/id-server2
Enter old passphrase:
```

```
Key has comment 'backup server2'
Enter new passphrase (empty for no passphrase):
Enter same passphrase again: passphrase
Your identification has been saved with the new passphrase.
```

Discussion

Passphrases are not recoverable. If you lose a passphrase, your only option is to create a new key with a new passphrase.

See Also

- OpenSSH
- *man 1 ssh_*
- *man 1 ssh-keygen*

12.10 Automatic Passphrase Management with Keychain

Problem

You want something to remember your private key passphrases for you, and use them as needed.

Solution

The Keychain utility was made for this. Install the *keychain* package, then copy the lines in the following example into your *.bashrc* file.

In the following example, you want access to *server1*, *server2*, and *server3* without entering your passphrases every time you log in. Copy these lines, except using your own key names:

```
keychain ~/.ssh/id-server1 ~/.ssh/id-server2 \
 ~/.ssh/id-server3 . ~/.keychain/$HOSTNAME-sh
```

Keychain keeps your private keys available until you shut down, so you must enter your passphrases every time you start up your system.

When you boot to a graphical environment, you may not be prompted to enter your passphrases. Try opening a terminal, and if you still don't see a Keychain prompt for your passphrases, you must enter a Linux console. Press Ctrl-Alt-F2 and log in. After logging in, you should see something like this:

```
* keychain 2.8.5 ~ http://www.funtoo.org
 * Found existing ssh-agent: 2016
 * Adding 3 ssh key(s): /home/duchess/.ssh/id-server1
```

```
/home/duchess/.ssh/id-server2 /home/duchess/.ssh/id-server3
Enter passphrase for /home/duchess/.ssh/id-server1:
Enter passphrase for /home/duchess/.ssh/id-server2:
Enter passphrase for /home/duchess/.ssh/id-server3:
 * ssh-add: Identities added: /home/duchess/.ssh/id-server1
/home/duchess/.ssh/id-server2 /home/duchess/.ssh/id-server3
```

Discussion

The leading dot in . *~/.keychain/$HOSTNAME-sh* is short for *source*, meaning use the named file.

$HOSTNAME tells Keychain to look in the user's environment variables to fetch their hostname. You can see this for yourself:

```
$ echo $HOSTNAME
pc
```

Keychain is a manager for both *ssh-agent* and *gpg-agent*, caching your SSH and GPG passphrases for as long as your computer is powered on. You can log out and log back in, and will have to reenter your passphrases only after a restart.

A good alternative is *gnome-keyring*, which runs in graphical environments. This provides a graphical interface for viewing and managing SSH and GPG keys, and it also includes a password manager. This appears as "Passwords and Keys" on most systems. Its has two disadvantage: it's not suitable to use on headless systems, and it does not make passphrases available to cron (see Recipe 12.11.)

See Also

- Funtoo Keychain

12.11 Using Keychain to Make Passphrases Available to Cron

Problem

You need to use cron to automate tasks, such as running rsync backups to a remote host. But no matter what you try, you get nothing for your troubles but failed backups with authentication errors.

Solution

To configure Keychain to manage your private keys for cron jobs, create a script for cron to use. The following example is for an rsync backup, and the script is named *duchess-backup-server1*:

```
#!/bin/bash
source $HOME/.keychain/${HOSTNAME}-sh
/usr/bin/rsync -ae "ssh -i /home/duchess/.ssh/id-server3" /home/duchess/ \
duchess@server1:/backups/
```

Make this script executable with *chmod*:

```
$ chmod +x duchess-backup-server1
```

This example adds a line to your crontab to run the script every night at 10:15 P.M.:

```
15 22 * * * /home/duchess/duchess-backup-server1
```

Discussion

In the example script, the line starting with */usr/bin/rsync* must be all on a single line.

Cron runs in its own special limited environment and needs Keychain to provide the required keys and environment variables.

See Also

- *man 1 crontab*
- Funtoo Keychain

12.12 Tunneling an X Session Securely over SSH

Problem

You want to run graphical applications from the remote host. You know that the X Window System has built-in networking abilities, but it sends all traffic in cleartext, which is insecure, and you want to do this safely.

Solution

Tunneling X over SSH requires no additional software. First, use these commands to see if your client machine is running the X11 or Wayland protocol. The following examples show both results:

```
$ echo $XDG_SESSION_TYPE
x11
$ echo $XDG_SESSION_TYPE
```

```
wayland
$ loginctl show-session "$XDG_SESSION_ID" -p Type
Type=x11
$ loginctl show-session "$XDG_SESSION_ID" -p Type
Type=wayland
```

loginctl is part of systemd.

If you are running Wayland, you cannot tunnel it over SSH because it does not have networking support.

If your system is using X11, configure X11 forwarding in */etc/ssh/sshd_config* on the remote machine:

```
X11Forwarding yes
```

The following example tunnels X over SSH, using the *-Y* option:

```
duchess@pc:~$ ssh -Yi id-server1 duchess@server1
Last login: Thu Jul  9 09:26:09 2021 from 192.168.43.80
Have a lot of fun..
duchess@server1:~$
```

Now you can run graphical applications, though only one at a time, like the game in Figure 12-1:

```
duchess@server1:~$ kmahjongg
```

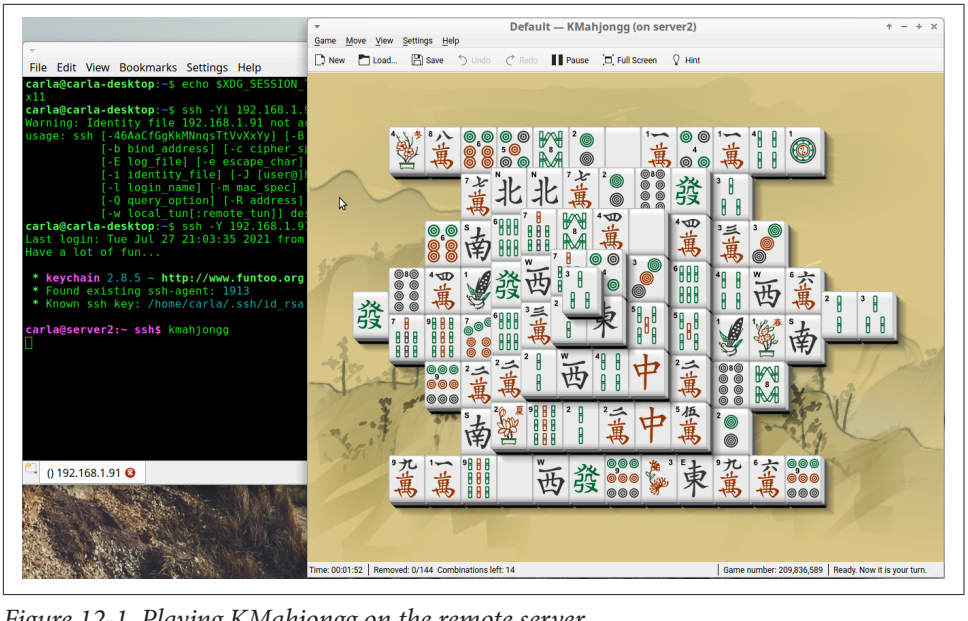


Figure 12-1. Playing KMahjongg on the remote server

Discussion

The X server runs with the offset specified in */etc/ssh/sshd.conf*, *X11DisplayOffset 10*. This avoids colliding with existing X sessions. Your regular local X session is :0.0, so your first remote X session is :10.0. You can see this with your own eyes. Run the following commands on your local machine. The first one is at your local command prompt:

```
duchess@pc:~$ echo $DISPLAY
:0.0
```

The second example is at your SSH command prompt:

```
duchess@server1:~ssh $ echo $DISPLAY
localhost:10.0
```

The remote system only needs to be powered on. You don't need any local users to be logged in, and you don't even need X to be running. X needs to be running only on the client PC.

See Also

- *man 1 sshd*
- *man 1 ssh_config*

12.13 Opening an SSH Session and Running a Command in One Line

Problem

You have a single command to run on the remote machine, and you think it would be nice to run it without logging in and running the command, and then logging out. After all, is it not true that laziness is a virtue for system administrators?

Solution

OpenSSH can do this. This example shows how to restart Postfix:

```
$ ssh mailadmin@server2.example.com sudo systemctl restart postfix
```

You'll be asked for a *sudo* password, but you will still save one whole step.

This shows how to open a quick game of GNOME Sudoku, which requires the X Window System:

```
$ ssh -Y duchess@laptop /usr/games/gnome-sudoku
```

Discussion

Another way to do this is with public key authentication for the root user, so you don't have to invoke *sudo* (Recipe 12.7).

See Also

- *man 1 ssh*

12.14 Mounting Entire Remote Filesystems with sshfs

Problem

OpenSSH is fast and efficient, and even tunneling X applications over OpenSSH isn't too laggy. But you want a faster way to edit a number of remote files without running a graphical file manager over SSH.

Solution

sshfs is the tool for you. *sshfs* is for mounting an entire remote filesystem, and then accessing it just like a local filesystem, without the hassles of setting up an NFS or Samba server.

Install the *sshfs* package, which should also install FUSE, the Filesystem in Userspace. You need a local directory that you have write permissions for as your mountpoint:

```
duchess@pc:~$ mkdir sshfs
```

Then mount your chosen remote directory in your local *sshfs* directory. This example mounts the home directory for *duchess@server2* in the *sshfs* directory at *duchess@pc*:

```
duchess@pc:~$ sshfs duchess@server2: sshfs/
```

The remote filesystem is just as accessible as your local filesystems:

```
duchess@pc:~$ ls sshfs
Desktop
Documents
Downloads
[...]
```

Access these files from the command line or with your graphical file manager, just like your local files.

Your command prompt will not change to the remote prompt.

When you're finished, unmount the remote filesystem:

```
duchess@pc:~$ fusermount -u sshfs/
```

That mounts Duchess's entire home directory. Specify a subdirectory instead:

```
duchess@pc:~$ sshfs duchess@server2:/home/duchess/arias sshfs/
```

You cannot use the tilde, ~, as a shortcut for */home/user* because *sshfs* does not support it.

If your network connection is not reliable, tell *sshfs* to automatically reconnect after an interruption:

```
duchess@pc:~$ sshfs duchess@server2:/home/duchess/arias sshfs/ -o reconnect
```

Discussion

Users who are new to *sshfs* always ask these questions: why not just run X over SSH, or why not just use NFS? The answers are: it is faster than running X over SSH, it is easier to set up than NFS, and you may use NFS, Samba, or whatever your heart desires.

See Also

- *man 1 sshfs*

12.15 Customizing the Bash Prompt for SSH

Problem

Sure, you know that the prompt changes to display the remote hostname when you're logged in via SSH. But it's just a plain prompt, and it's easy to make mistakes, so you want a customized, colorful prompt to indicate when you have an active SSH login.

Solution

Customize the Bash prompt on the remote machines. This example turns the prompt purple and adds "ssh" to it.

Copy these lines into the *.bashrc* file for the remote account you want to log in to:

```
if [ -n "$SSH_CLIENT" ]; then text=" ssh"
fi
export PS1='\[\e[0;36m\]\u@\h:\w${text}$\[\e[0m\] '
```

When you log in to this machine, the prompt will look like what's shown in Figure 12-2.

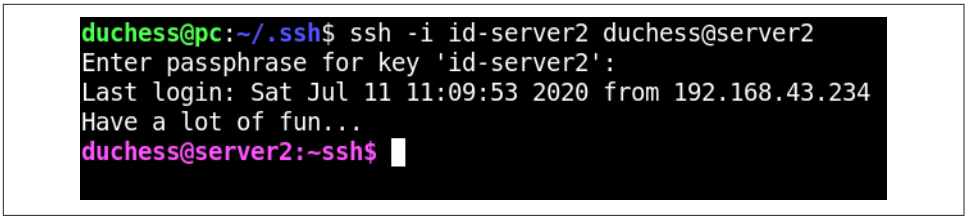


Figure 12-2. A customized SSH prompt

Only the prompt is purple, and all the other text will be your normal shell colors.

Discussion

Customizing the Bash prompt is practically a book topic in itself. The example in this recipe can be edited to suit your preferences. You don't have to use the term "ssh" or name the variable "text"; these can be anything you like. You could say "super duper encrypted session" and name your variable "sekkret-squirl" if you want.

[\e[0;31m\] is the code block that determines the text color. All you have to do is change the numbers to change the colors.

[\e[0m\] turns off the custom colors, so that your commands and command output will return to the normal shell colors. Here are the color codes:

- Black 0;30
- Blue 0;34
- Green 0;32
- Cyan 0;36
- Red 0;31
- Purple 0;35
- Brown 0;33
- Light Gray 0;37
- Dark Gray 1;30
- Light Blue 1;34
- Light Green 1;32
- Light Cyan 1;36
- Light Red 1;31
- Light Purple 1;35
- Yellow 1;33
- White 1;37

This customization works by checking for the presence of the *SSH_CLIENT* environment variable, which is present only when there is an active SSH connection. You can see this for yourself on the remote host:

```
$ echo $SSH_CLIENT
192.168.43.234 51414 22
```

Then Bash knows to use the custom SSH prompt instead of the default prompt. When you run this command on a machine without any active SSH sessions, it returns an empty line.

See Also

- *man 1 bash*
- Bash Prompt HOWTO, Chapter 6

12.16 Listing Supported Encryption Algorithms

Problem

You have compliance rules to follow and need to know what encryption algorithms OpenSSH supports.

Solution

OpenSSH includes a command to query and list all supported algorithms, *ssh -Q <query_option>*. List them with the *help* option:

```
$ ssh -Q help
cipher
cipher-auth
compression
kex
kex-gss
key
key-cert
key-plain
key-sig
mac
protocol-version
sig
```

The following example lists the *sig* signature algorithms:

```
$ ssh -Q sig
ssh-ed25519
sk-ssh-ed25519@openssh.com
ssh-rsa
rsa-sha2-256
rsa-sha2-512
ssh-dss
ecdsa-sha2-nistp256
ecdsa-sha2-nistp384
ecdsa-sha2-nistp521
sk-ecdsa-sha2-nistp256@openssh.com
```

Discussion

The following list briefly describes each option:

- *cipher* lists supported symmetric ciphers.
- *cipher-auth* lists supported symmetric ciphers that also support authenticated encryption.
- *compression* lists supported compression types.
- *mac* lists supported message integrity codes. These protect your message's data integrity and its authenticity.
- *kex* lists key exchange algorithms.
- *kex-gss* lists GSSAPI (Generic Security Service Application Program Interface) key exchange algorithms.
- *key* lists key types.
- *key-cert* lists certificate key types.
- *key-plain* lists noncertificate key types.
- *key-sig* lists all key types and signature algorithms.
- *protocol-version* lists supported SSH protocol versions, which is only version 2 at the time of writing.
- *sig* lists supported signature algorithms.

See Also

- OpenSSH
- *Serious Cryptography* by Jean-Philippe Aumasson (No Starch Press)

CHAPTER 13

Secure Remote Access with OpenVPN

Open Virtual Private Network (OpenVPN) creates a TLS/SSL encrypted connection between two different networks at separate physical locations, like a branch office linked to a main office, or a remote worker logging in to the company network from home. This connection is called an encrypted *tunnel*, a secure transport protecting your connection from the big bad internet. OpenVPN is dependent on OpenSSL, so having OpenSSL knowledge is helpful.



If you are already familiar with OpenVPN, you can probably skip ahead to Recipes 13.5, 13.6, and 13.7 to review creating your encryption certificates and client and server configuration. If you are new to VPNs, try each recipe in sequence. Take your time; VPNs are complicated and finicky. Do a lot of testing before deploying to production systems.

OpenVPN Overview

A VPN is a secure extension of your network that makes all the same services available to remote workers that local users have, so the remote users' experience is the same as for users physically present at your location. They can access your local web servers, email, file shares, chat servers, video conferencing apps, internal wikis, everything that you have walled off from the outside world and is available only to users inside your network. A VPN is not like SSH, which connects individual computers. A VPN links etworks and individual hosts to networks.

In this chapter you will learn how to set up an OpenVPN server, configure clients, and create and manage a proper public key infrastructure (PKI) for authentication and encryption. Your server will authenticate and protect all manner of clients: Linux, macOS, Windows PCs, Android, and iOS devices.

OpenVPN is an open source project, with both free downloads and commercial options. The free-of-cost server and client is the *openvpn* package, which is available on all Linux distros and downloads at OpenVPN Community Downloads. Commercial options include OpenVPN Access Server, which is a premises server with additional management tools and cloud options. Hosted personal plans require installing only the client and provide access to a global network of OpenVPN servers.

A true VPN is strong because it trusts no one and requires authenticated endpoints, where server and client authenticate to each other. Most commercial TLS/SSL VPNs do not do this, but instead trust all clients, like shopping sites do. This is more flexible and allows users to log in from anywhere, using any device. It is convenient not to have to install and configure client software and copy encryption keys. But for your internal network that is shortsighted—the last thing you need is users logging in from random PCs or smartphones infected with keyloggers and spyware, and then given a warm welcome into your LAN.

Certificate Authority

A certificate authority (CA) is the most important part of running an OpenVPN server. A CA issues digital certificates and certifies ownership of public keys. Click the little padlock in a web browser to see the public certificate for a website, and which CA signed it. A CA is a trusted authority, and that is why so many sites use commercial CAs. Self-signed certificates, like the ones we are creating in this chapter, are fine to use inside your organization. Customer-facing sites should use commercial CAs. Using a CA saves you from the hassle of keeping copies of client certificates on your OpenVPN server; all the server needs to know is that the client certificate is authenticated by your CA.

SSL Versus TLS

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) are cryptographic protocols. TLS evolved from SSL. All versions of SSL are deprecated, as are TLS 1.0 and TLS 1.1. Use TLS 1.2 or 1.3, and disable all the others (Recipe 13.10). Older versions are deprecated because of security flaws, so don't let anyone talk you into supporting deprecated versions.

TUN/TAP

The *TUN* and *TAP* devices are virtual network interfaces. These are built into the Linux kernel, and you should not have to do anything to make them available. The *TUN* device is for routed networks, and the *TAP* device is for bridged networks. Your server and client configuration files specify which one to use.



Good Security Takes Work

Good security requires ongoing study and maintenance. This chapter aims to show you how to set up a strong VPN that is reasonably user-friendly. There are many additional methods for making your VPN even stronger, such as client certificates with short lifetimes, additional authentications, hardware devices, SELinux, chroot jails, short password timeouts, and many more. If you require super-high security, please consult expert professionals.

13.1 Installing OpenVPN, Server and Client

Problem

You need to know how to install openVPN.

Solution

The OpenVPN website supplies both the community open source OpenVPN and the commercial OpenVPN Access Server. The community OpenVPN is free of cost and open source. This chapter covers the community OpenVPN.

On Linux, install the *openvpn* package. (As always, verify the package name on your particular Linux.) Get the most current version you can, from 2.4.5 and up. This provides both server and client. Source tarballs and Windows installers are available from OpenVPN Community Downloads.

For your clients, you could try the free OpenVPN Access Clients, which are available for Linux, macOS, Android, iOS, and Windows. These are designed for the commercial OpenVPN Access Server and also work with the community OpenVPN server.

You can also find the community OpenVPN client for Android in the Google Play Store.

See Recipe 13.9 to learn how to use the *.ovpn* inline file format for easier client configuration.

Discussion

On Linux, OpenVPN must be installed on your OpenVPN server and on all clients. The OpenVPN package provides both client and server functionality.

Ubuntu, Fedora, and openSUSE include additional packages that provide integration with NetworkManager, which makes managing, connecting, and disconnecting to VPNs nice and easy.

NetworkManager-openvpn (Fedora, openSUSE) and *network-manager-openvpn* (Ubuntu) integrate OpenVPN with Network Manager. If you are using the GNOME environment (such as GNOME, Xfce, Cinnamon, or Mate), you also need *NetworkManager-openvpn-gnome* (openSUSE, Fedora), or *network-manager-openvpn* (Ubuntu).

OpenVPN Access Server is a free download, and you may connect up to two clients at the same time without purchasing a license. It comes with additional features, such as a web administration interface and automatic configurations with the free OpenVPN Access Client. If you start with the community OpenVPN and then decide to migrate to OpenVPN Access Server, everything you learned for the community OpenVPN server applies to Access Server as well.

See Also

- EasyRSA
- OpenVPN documentation
- *man 8 openvpn*
- OpenSSL Cookbook

13.2 Setting Up a Simple Connection Test

Problem

You want to run the simplest OpenVPN connection test to get an idea of how it works and to verify connectivity.

Solution

The following simple test creates an unencrypted tunnel between two Linux computers that are on the same network. OpenVPN must be installed on both of them. First verify that the OpenVPN daemon is not running on either host, and if it is, stop it:

```
$ systemctl status openvpn@.openvpn1.service
● openvpn.service - OpenVPN service
     Loaded: loaded (/lib/systemd/system/openvpn.service; enabled; vendor prese>
     Active: active (exited) since Sun 2021-01-10 13:43:18 PST; 33min ago
[...]
$ sudo systemctl stop openvpn@.openvpn1.service
```



Use a Different Subnet for Your VPN

Use a different subnet for your OpenVPN tunnel; for example, *host1* and *host2* are on 192.168.43.0/24, so the example uses the 10.0.0.0/24 private address space for the VPN tunnel.

In the following example, the two computers are named *host1* and *host2*. The first example creates a VPN tunnel from *host1* to *host2*:

```
[madmax@host1 ~]$ sudo openvpn  --remote host2 --dev tun0 --ifconfig 10.0.0.1 |
10.0.0.2
Sat Jan  9 14:40:34 2021 disabling NCP mode (--ncp-disable) because not in P2MP
  client or server mode
Sat Jan  9 14:40:34 2021 OpenVPN 2.4.8 x86_64-redhat-linux-gnu [SSL (OpenSSL)]
  [LZO] [LZ4] [EPOLL] [PKCS11] [MH/PKTINFO] [AEAD] built on Jan 29 2020
Sat Jan  9 14:40:34 2021 library versions: OpenSSL 1.1.1d FIPS  10 Sep 2019,
  LZO 2.10
Sat Jan  9 14:40:34 2021 ******* WARNING *******: All encryption and
  authentication features disabled -- All data will be tunnelled as clear text
  and will not be protected against man-in-the-middle changes. PLEASE DO
  RECONSIDER THIS CONFIGURATION!
Sat Jan  9 14:40:34 2021 TUN/TAP device tun0 opened
Sat Jan  9 14:40:34 2021 /sbin/ip link set dev tun0 up mtu 1500
Sat Jan  9 14:40:34 2021 /sbin/ip addr add dev tun0 local 10.0.0.1 peer 10.0.0.2
Sat Jan  9 14:40:34 2021 TCP/UDP: Preserving recently used remote address:
  [AF_INET]192.168.122.239:1194
Sat Jan  9 14:40:34 2021 UDP link local (bound): [AF_INET][undef]:1194
Sat Jan  9 14:40:34 2021 UDP link remote: [AF_INET]192.168.122.239:1194
```

This example creates a link from *host2* to *host1*:

```
[stash@host2 ~]$ sudo openvpn --remote host1 --dev tun0 --ifconfig 10.0.0.2 |
10.0.0.1
Sat Jan  9 14:50:53 2021 disabling NCP mode (--ncp-disable) because not in P2MP
  client or server mode
Sat Jan  9 14:50:53 2021 OpenVPN 2.4.7 x86_64-pc-linux-gnu [SSL (OpenSSL)] [LZO]
  [LZ4] [EPOLL] [PKCS11] [MH/PKTINFO] [AEAD] built on Sep  5 2019
Sat Jan  9 14:50:53 2021 library versions: OpenSSL 1.1.1f  31 Mar 2020, LZO 2.10
Sat Jan  9 14:50:53 2021 ******* WARNING *******: All encryption and
  authentication features disabled -- All data will be tunnelled as clear text
  and will not be protected against man-in-the-middle changes. PLEASE DO
  RECONSIDER THIS CONFIGURATION!
Sat Jan  9 14:50:53 2021 TUN/TAP device tun0 opened
Sat Jan  9 14:50:53 2021 /sbin/ip link set dev tun0 up mtu 1500
Sat Jan  9 14:50:53 2021 /sbin/ip addr add dev tun0 local 10.0.0.2 peer 10.0.0.1
Sat Jan  9 14:50:53 2021 TCP/UDP: Preserving recently used remote address:
  [AF_INET]192.168.122.52:1194
Sat Jan  9 14:50:53 2021 UDP link local (bound): [AF_INET][undef]:1194
Sat Jan  9 14:50:53 2021 UDP link remote: [AF_INET]192.168.122.52:1194
Sat Jan  9 14:51:03 2021 Peer Connection Initiated with
  [AF_INET]192.168.122.52:1194
Sat Jan  9 14:51:04 2021 WARNING: this configuration may cache passwords in
  memory -- use the auth-nocache option to prevent this
Sat Jan  9 14:51:04 2021 Initialization Sequence Completed
```

You have a successful connection when both hosts show the "Initialization Sequence Completed" message. Test your connections by pinging through the *tun0* interface on both hosts:

```
[madmax@host1 ~]$ ping -I tun0 10.0.0.2
PING 10.0.0.2 (10.0.0.2) from 10.0.0.1 tun0: 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.515 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.436 ms

[stash@host2 ~]$ ping -I tun0 10.0.0.1
PING 10.0.0.1 (10.0.0.1) from 10.0.0.2 tun0: 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.592 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.534 ms
```

Press Ctrl-C on each host to stop ping, and again to close the tunnels.

Discussion

This simple test illustrates how OpenVPN works. It creates a virtual network interface, *tun0* on both hosts, then routes network traffic through this interface. This simple test does not create an encrypted connection, as you can see from the "******* WARNING *******: All encryption and authentication features disabled" message in your command output.

See Also

- EasyRSA
- OpenVPN documentation
- systemd.unit
- *man 8 openvpn*
- OpenSSL Cookbook

13.3 Setting Up Easy Encryption with Static Keys

Problem

You want an easy way to create and manage encryption for OpenVPN.

Solution

The easiest method is to use shared static keys. Shared static keys are useful for testing, but they are not adequate for production systems. (See the Discussion to learn about their shortcomings.) In this recipe you will learn how to create and share static keys, and how to create simple server and client configuration files.

Follow these steps:

1. Create and distribute a shared static key between two hosts.
2. Create server and client configuration files.
3. Start OpenVPN on both hosts, referencing their configuration files.

In the following examples, the OpenVPN server is on *server1*, the client is *client1*, and the new key is *myvpn.key*. You may name your keys whatever you want.

Create a new directory on the OpenVPN server to store keys, then create a new static key:

```
$ sudo mkdir /etc/openvpn/keys
$ sudo openvpn --genkey --secret myvpn.key
```

Copy the key to the client machine:

```
$ scp myvpn.key client1:/etc/openvpn/keys/
Password:
myvpn.key                         100%  636   142.7KB/s   00:00
```

Create the server configuration file. The example is */etc/openvpn/server1.conf*, and you can call yours anything you like. Use a different subnet for your OpenVPN tunnel; for example, *server1* and *client1* are on 192.168.43.0/24, so the example uses the 10.0.0.0/24 private address space for the VPN tunnel. The server's *tun* address is 10.0.0.1:

```
# server1.conf
dev tun
ifconfig 10.0.0.1 10.0.0.2
secret /etc/openvpn/keys/myvpn.key
local 192.168.43.184
```

local is the LAN IP address of the server.

Create the client configuration file on the client machine. The client's *tun* address is 10.0.0.2:

```
# client1.conf
dev tun
ifconfig 10.0.0.2 10.0.0.1
secret /etc/openvpn/keys/myvpn.key
remote 192.168.43.184
```

Make sure the OpenVPN daemon is not running on the server or client:

```
$ sudo systemctl stop openvpn
```

Start OpenVPN on the server and client:

```
[server1 ~] $ sudo openvpn /etc/openvpn/server1.conf

[client1 ~] $ sudo openvpn /etc/openvpn/client1.conf
```

When you see "Initialization Sequence Completed" on both hosts, you have established a connection. Ping each host over the *tun* virtual network interface:

```
[server1 ~] $ ping -I tun0 10.0.0.1
[client1 ~] $ ping -I tun0 10.0.0.2
```

Press Ctrl-C on both hosts to close the connection.

Discussion

If you see "WARNING: INSECURE cipher with block size less than 128 bit (64 bit). This allows attacks like SWEET32. Mitigate by using a cipher with a larger block size (e.g. AES-256-CBC)" in your command output, correct it with the following entry in both the server and client configuration files:

```
cipher AES-256-CBC
```

The biggest problem with using static keys is that you lose perfect forward secrecy because your static key never changes. If an attacker found a way to sniff and capture your network traffic, and then captured and cracked your encryption key, the attacker could then decrypt everything they capture, past and future. OpenVPN's PKI uses a complex process that generates session keys, which are not persistent but change regularly. So, at best, a successful attacker can decrypt one session's worth of traffic at a time, and then has to start over.

Another drawback is you need a different key for each client, and a copy of each client key on the server. Managing multiple clients is less work and more secure with a proper PKI.

See Also

- EasyRSA
- OpenVPN documentation
- systemd.unit
- *man 8 openvpn*
- OpenSSL Cookbook

13.4 Installing EasyRSA to Manage Your PKI

Problem

You are going to use EasyRSA to create and manage your public key infrastructure (PKI), and you want to install and set it up correctly.

Solution

Your PKI can be anywhere, it does not have to be on the OpenVPN server. You will create server and client certificates on the PKI, then copy them to their respective hosts.

You can install the *easy-rsa* package or fetch the freshest release from EasyRSA Releases on GitHub.

Fedora and Ubuntu stuff all the EasyRSA files into */usr/share/*. This is not a good working directory, and it is overwritten by system updates. Create a new directory that you control and does not need root permissions, like our fine example user Duchess who creates */home/duchess/mypki*:

```
~$ mkdir mypki
```

On Fedora and Ubuntu Linux, copy the */usr/share/easy-rsa* directory to your new directory:

```
~$ sudo cp -r /usr/share/easy-rsa mypki
```

This creates *mypki/easyrsa*. Check your permissions; you should be the owner and group owner of everything in your directory.

openSUSE does a proper installation with configuration files in */etc/easy-rsa*, the *easy-rsa* command in */usr/bin*, and the documentation and license files in */usr/share/*. You don't have to move anything or worry about permissions.

Discussion

You should not need root permissions to create and manage your PKI. You may put it wherever you want, and it should be separate from your OpenVPN configuration, either in a separate directory or on a separate machine. The good OpenVPN people recommend putting it on a well-protected machine that is not exposed to the internet.

See Also

- EasyRSA
- OpenVPN documentation
- systemd.unit
- *man 8 openvpn*
- OpenSSL Cookbook

13.5 Creating a PKI

Problem

You installed EasyRSA (Recipe 13.4), and now you want to know how to set up a proper public key infrastructure (PKI).

Solution

A proper PKI is essential to running an OpenVPN server safely. In this recipe we will use EasyRSA to create a PKI, which simplifies the process considerably over using the *openssl* command. Creating a PKI involves these steps:

1. Create your own Certificate Authority (CA) certificate to sign server and client certificates. This should be in a separate directory from your OpenVPN server configuration, or on a separate machine.
2. Create and sign an OpenVPN server certificate.
3. Create and sign client certificates.
4. Copy the server certificates and the client certificates to */etc/openvpn/keys* on their respective machines. (You may create a different directory than *keys*.)

In the following examples, all the commands are run from the */home/duchess/mypki/* directory.

Change to your PKI directory and run the command to initiate a new PKI:

```
~$ cd mypki
~/mypki $ easyrsa init-pki

init-pki complete; you may now create a CA or requests.
Your newly created PKI dir is: /home/duchess/mypki/pki
```

This creates an empty structure for your new PKI. Next, build your new CA. The CA creates and signs server and client certificates. Protect it with a strong passphrase, and create the Common Name you want for your new CA:

```
~/mypki $ easyrsa build-ca
[...]
Enter New CA Key Passphrase:passphrase
Re-Enter New CA Key Passphrase:passphrase
[...]
Common Name (eg: your user, host, or server name) [Easy-RSA CA]: vpnserver1
[...]
CA creation complete and you may now import and sign cert requests.
Your new CA certificate file for publishing is at:
/home/duchess/mypki/pki/ca.crt
```



If you see a "RAND_load_file:Cannot open file:crypto/rand/randfile.c:98:Filename=/mypki/pki/.rnd" message, ignore it, as it is meaningless. You can make it go away by finding *openssl-easyrsa.cnf* and commenting out the *RANDFILE* line at the beginning.

Generate a keypair and certificate signing request for your OpenVPN server. It is customary to not put a passphrase on the server's private key. You may protect yours with a passphrase if you wish by omitting the *nopass* option. A passphrase provides strong protection, but it means entering the passphrase every time you restart your server:

```
~/mypki $ easyrsa gen-req vpnserver1 nopass

Using SSL: openssl OpenSSL 1.1.1d  10 Sep 2019
Generating a RSA private key
.............................+++++
.....................................................................+++++++
writing new private key to '/home/duchess/mypki/pki/private/vpnserver1.key.NYjr5y
c9kj'
[...]
Common Name (eg: your user, host, or server name) [vpnserver1]:

Keypair and certificate request completed. Your files are:
req: /home/duchess/mypki/pki/reqs/vpnserver1.req
key: /home/duchess/mypki/pki/private/vpnserver1.key
```

Generate a keypair and certificate signing request for a client. Client private keys should have passwords, especially on mobile clients:

```
~/mypki $ easyrsa gen-req vpnclient1

Using SSL: openssl OpenSSL 1.1.1d  10 Sep 2019
Generating a RSA private key
................+++++
.....................................................................+++++
writing new private key to '/home/duchess/mypki/pki/private/vpnclient1.key.bicpOc
EC5S'
Enter PEM pass phrase:passphrase
Verifying - Enter PEM pass phrase:passphrase
[...]
Common Name (eg: your user, host, or server name) [vpnclient1]:

Keypair and certificate request completed. Your files are:
req: /home/duchess/mypki/pki/reqs/vpnclient1.req
key: /home/duchess/mypki/pki/private/vpnclient1.key
```

Sign the requests, using their Common Names. Use only their names; if you enter their paths it will cause an error:

```
~/mypki $ easyrsa sign-req server vpnserver1
Using SSL: openssl OpenSSL 1.1.1d  10 Sep 2019
```

```
You are about to sign the following certificate.
Please check over the details shown below for accuracy. Note that this request
has not been cryptographically verified. Please be sure it came from a trusted
source or that you have verified the request checksum with the sender.

Request subject, to be signed as a server certificate for 1080 days:

subject=
    commonName                = vpnserver1

Type the word 'yes' to continue, or any other input to abort.
  Confirm request details: yes
Using configuration from /home/duchess/mypki/pki/safessl-easyrsa.cnf
Enter pass phrase for /home/duchess/mypki/pki/private/ca.key:
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'vpnserver1'
Certificate is to be certified until Jan 27 20:09:12 2024 GMT (1080 days)

Write out database with 1 new entries
Data Base Updated

Certificate created at: /home/duchess/mypki/pki/issued/vpnserver1.crt

mypki $ easyrsa sign-req client vpnclient1
[...]
Certificate created at: /home/duchess/mypki/pki/issued/vpnclient1.crt
```

Generate the Diffie-Hellman parameters for the server; this takes a minute or two. This command must be run on your OpenVPN server:

```
$ easyrsa gen-dh
Using SSL: openssl OpenSSL 1.1.1d  10 Sep 2019
Generating DH parameters, 2048 bit long safe prime, generator 2
This is going to take a long time
.....................................+...........
..........+.....................................
[...]
DH parameters of size 2048 created at /home/duchess/mypki/pki/dh.pem
```

Create a Hash-based Message Authentication Code (HMAC) key, also on your server:

```
$ openvpn --genkey --secret ta.key
```

Copy *vpnclient1.key*, *vpnclient1.crt*, *ca.crt*, and *ta.key* to */etc/openvpn/keys* on *client1*.

Copy *vpnserver1.key*, *vpnserver1.crt*, *ca.crt*, *dh.pem*, and *ta.key* to */etc/openvpn/keys* on *server1*.

You may delete all of the **.req* files after you have signed the certificate signing requests.

Table 13-1 should help you remember which files go where.

Table 13-1. Server and client key location

Name	Location	Public	Private
ca.crt	server & clients	X	
ca.key	PKI machine		X
ta.key	server & clients		X
dh.pem	server	X	
server.crt	server	X	
server.key	server		X
client1.crt	client1	X	
client1.key	client1		X
client2.crt	client2	X	
client2.key	client2		X

Discussion

What's this Diffie-Hellman stuff? It is the encryption mechanism that allows two hosts to create and share a secret key. Once the OpenVPN client and server authenticate to each other, additional send and receive keys are generated to encrypt the session.

HMAC calculates a message authentication code. HMAC verifies the integrity and authenticity of a message.

easyrsa init-pki creates a new PKI, and you may also run it to cleanly remove and rebuild an existing PKI.

You may set up your PKI anywhere, and the good OpenVPN people recommend putting it on a machine that is not exposed to the internet and is well-protected from anyone who should not be messing with your PKI. If your CA is compromised, it is easy for an attacker to infiltrate your network. Obviously, you must have a secure method of distributing these files: USB stick, the *scp* command, an encrypted tarball downloaded from a secure server or emailed to your users.

Take a look in your PKI directory to see how all these items are organized.

```
~/mypki $ ls */*
pki/ca.crt                pki/index.txt             pki/index.txt.old
pki/serial                pki/dh.pem                pki/index.txt.attr
pki/openssl-easyrsa.cnf   pki/serial.old            pki/extensions.temp
pki/index.txt.attr.old    pki/safessl-easyrsa.cnf   pki/ta.key

pki/certs_by_serial:
4954C26DB44106B20F1B9DA17CE515E5.pem  DA68CBE53E30923C9BCC3B9F1C5C9011.pem
```

```
pki/issued:
vpnclient1.crt  vpnserver1.crt

pki/private:
ca.key  vpnclient1.key  vpnserver1.key

pki/renewed:
certs_by_serial  private_by_serial  reqs_by_serial

pki/reqs:
vpnclient1.req  vpnserver1.req

pki/revoked:
certs_by_serial  private_by_serial  reqs_by_serial
```

Signing requests have a *.req* file extension, public keys *.crt*, and private keys *.key*. Keys always come in pairs, public and private.



Public Keys Encrypt, Private Keys Decrypt

Public keys encrypt, private keys decrypt. Private keys must be protected and never shared. Public keys are meant to be shared.

Click on any signed certificate in your file manager to see something like Figure 13-1. This provides a wealth of information: the CA that signed it, expiration date, serial number, fingerprint, signature, and lots more.

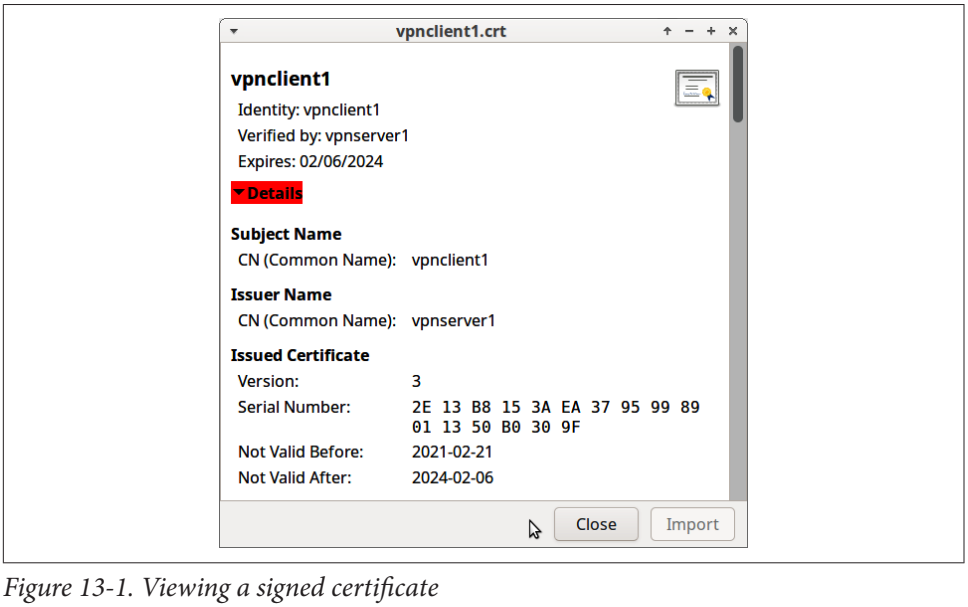


Figure 13-1. Viewing a signed certificate

Or use the *openssl* command to read it:

```
$ openssl x509 -noout -text -in vpnserver1.crt
```

A certificate is a request signed by a CA, and it contains the public key and the CA's digital signature. A request contains a public key and the digital signature from the corresponding private key. You can see all of this by comparing them.

EasyRSA was originally part of OpenVPN and then spun off as a separate project. If you are used to managing PKIs with OpenSSL, you will appreciate how EasyRSA has streamlined the process.

See Also

- EasyRSA
- OpenVPN documentation
- *man 8 openvpn*
- OpenSSL Cookbook

13.6 Customizing EasyRSA Default Options

Problem

The default settings for EasyRSA are not what you want, and you want to know how to change them.

Solution

Look for your *vars.example* file, which is part of EasyRSA. Save a copy of this file as *vars* in your PKI directory, which in the examples in this chapter is */home/duchess/mypki/pki/*. The *vars* file defines your default settings for creating and signing certificates.

This file is well commented. Make your edits below the `# DO YOUR EDITS BELOW THIS POINT` line. Everything prefaced with *set_var* is editable. Uncomment everything you change.

For example, the default configuration uses just the Common Name, and not a full *org* configuration. The following example creates a traditional *org* configuration:

```
set_var EASYRSA_DN      "org"

set_var EASYRSA_REQ_COUNTRY    "US"
set_var EASYRSA_REQ_PROVINCE   "Oregon"
set_var EASYRSA_REQ_CITY       "Walla Walla"
```

```
set_var EASYRSA_REQ_ORG         "MyCo"
set_var EASYRSA_REQ_EMAIL       "me@example.com"
set_var EASYRSA_REQ_OU          "MyOU"
```

When you use the *org* configuration, remember to enter your Common Name when you run *easyrsa build-ca*, or you will be stuck with the default *Easy-RSA CA*:

```
Common Name (eg: your user, host, or server name) [Easy-RSA CA]:myCN
```

Discussion

Use *cn* or *org* according to your own policies and preferences; it makes no difference to your server operations.

See Recipe 13.10 to learn how to harden your server.

See Also

- EasyRSA
- OpenVPN documentation
- *man 8 openvpn*
- OpenSSL Cookbook

13.7 Creating and Testing Server and Client Configurations

Problem

Now that you have a nice stout PKI, you want to know how to configure your OpenVPN server and clients.

Solution

In this recipe we will set up a simple test instance between two hosts on the same subnet, *server1* and *client1*. This is a good simple way to test your server configuration without having to hassle with routing and getting past your internet gateway.

The following example is a simple OpenVPN server configuration. Note that you can store your server keys anywhere on the server, as long you reference them correctly in your configuration file:

```
# vpnserver1.conf
port 1194
proto udp
dev tun
user nobody
group nobody
```

```
ca /etc/openvpn/keys/ca.crt
cert /etc/openvpn/keys/vpnserver1.crt
key /etc/openvpn/keys/vpnserver1.key
dh /etc/openvpn/keys/dh.pem
tls-auth /etc/openvpn/keys/ta.key 0

server 10.10.0.0 255.255.255.0
ifconfig-pool-persist ipp.txt
keepalive 10 120
persist-key
persist-tun
tls-server
remote-cert-tls client

status openvpn-status.log
verb 4
mute 20
explicit-exit-notify 1
```

An example client configuration:

```
# vpnclient1.conf
client
dev tun
proto udp
remote server1 1194

persist-key
persist-tun
resolv-retry infinite
nobind

user nobody
group nobody
tls-client
remote-cert-tls server
verb 4

ca /etc/openvpn/keys/ca.crt
cert /etc/openvpn/keys/vpnclient1.crt
key /etc/openvpn/keys/vpnclient1.key
tls-auth /etc/openvpn/keys/ta.key 1
```

Stop your OpenVPN server if it is running:

```
$ sudo systemctl stop openvpn@.openvpn1.service
```

Start OpenVPN on both hosts with the *openvpn* command:

```
$ sudo openvpn /etc/openvpn/vpnserver1.conf
Tue Feb 16 16:50:49 2021 us=265445 Current Parameter Settings:
Tue Feb 16 16:50:49 2021 us=265481   config = '/etc/openvpn/vpnserver1.conf'
```

```
[...]
Tue Feb 16 16:50:49 2021 us=270212 Initialization Sequence Completed

$ sudo openvpn /etc/openvpn/vpnclient1.conf
Tue Feb 16 16:56:22 2021 OpenVPN 2.4.3 x86_64-suse-linux-gnu [SSL (OpenSSL)]
[LZO] [LZ4] [EPOLL] [PKCS11] [MH/PKTINFO] [AEAD] built on Jun 20 2017
Tue Feb 16 16:56:22 2021 library versions: OpenSSL 1.1.1d  10 Sep 2019, LZO 2.10
Enter Private Key Password: *******
[...]
Tue Feb 16 16:56:26 2021 Initialization Sequence Completed
```

And there you have it, both configurations are correct and you have a successful connection. Press Ctrl-C on both hosts to stop.

Discussion

OpenVPN installs with a batch of example configurations in */usr/share/doc/openvpn/*. These are abundantly commented and are excellent references. There are dozens of options, but in real life you will use just a few of them. In the examples in this recipe, there are a few items of note.

`port 1194` is the default port, and `proto udp` is preferred over `proto tcp`. UDP is more secure, providing some protection from port scanning and denial-of-service attacks, and provides higher throughput and lower latency. TCP is useful when a remote user uses public networks with restrictive firewalls, such as hotels and coffee shops.

tls-auth /etc/openvpn/keys/ta.key must always have the 0 value on the server, and 1 on the clients. *tls-auth* enforces TLS-only connections.

verb 4 is the logging level. 1 is the lowest, 9 is the most verbose. Keep it at 4–6 until you are confident everything is set up correctly. When you start OpenVPN from the command line, you will see a lot of messages.

There are a lot of stale how-tos that recommend the *comp_lzo* option to enable compression. Don't bother, as it does not provide much benefit. Most traffic is not compressible since it is either already compressed or is encrypted and cannot be compressed. There is at least one vulnerability enabled by compression, VORACLE.

See Also

- The example configuration files in your installation
- EasyRSA
- OpenVPN documentation
- *man 8 openvpn*
- OpenSSL Cookbook

13.8 Controlling OpenVPN with systemctl

Problem

You want to manage the OpenVPN daemon like any other daemon, with *systemctl*, but you don't see an OpenVPN unit file. Or, you see an odd-looking unit file like *openvpn-server@.service*, and when you try to start it, it throws error messages.

Solution

The ampersand, @, creates a *parameterized* unit file. This means you can easily create multiple unit files for the same service by calling different configuration files. For example, suppose your server configuration file is */etc/openvpn/austin.conf*. Your unit file is *openvpn@austin.service*, created with *systemctl*:

```
$ sudo systemctl enable openvpn@austin
Created symlink /etc/systemd/system/multi-user.target.wants/openvpn@austin.service
→ /usr/lib/systemd/system/openvpn@.service.
Created symlink /etc/systemd/system/openvpn.target.wants/openvpn@austin.service
→ /usr/lib/systemd/system/openvpn@.service.
```

Note that you do not type the file extension of your OpenVPN *.conf* file. Now you can control your OpenVPN daemon with *systemctl*, just like any other service.

Discussion

This is a rather ingenious method that gives you the flexibility to create multiple configurations without having to write multiple unit files. You can "parameterize" any systemd unit file.

You can have multiple tunnels running at the same time on the same machine. Each configuration requires a different *tun* device, for example *tun0*, *tun1*, *tun2*, a different subnet for each tunnel, and a different UDP port. Control all of these tunnels with different configuration files and their corresponding parameterized unit files.

See Also

- EasyRSA
- OpenVPN documentation
- systemd.unit
- *man 8 openvpn*
- OpenSSL Cookbook

13.9 Distributing Client Configurations More Easily with .ovpn Files

Problem

Setting up clients is a fair bit of work, and you want to know if there is a faster way, one that your users can do themselves without a lot of help.

Solution

Bundle your client configurations and keys into single files with the *.ovpn* extension. All clients, Linux, Windows, macOS, iOS, and Android, can import these.

First create the users' certificates, then follow this template to create their *.ovpn* files. This example builds on the example in Recipe 13.7. Instead of linking to all their certificates, copy them into this file. All certificates are in plain text, so all you do is copy the BEGIN/END portions into the *.ovpn* file:

```
#vpnclient1.ovpn
client
dev tun
proto udp
remote server2 1194

persist-key
persist-tun
resolv-retry infinite
nobind

user nobody
group nobody
tls-client
remote-cert-tls server
verb 4

# ca.crt
<ca>
-----BEGIN CERTIFICATE-----
MIIDSDCCAjCgAwIBAgIUD2UxdEwgvhhr0zq5fAxIDIueB2EwDQYJKoZIhvcNAQEL
BQAwFTETMBEGA1UEAwwKdnBuc2VydmVyMTAeFw0yMTAyMjExODU1MjNaFw0zMTAy
MTkxODU1MjNaMBUxEzARBgNVBAMMCnZwbnNlcnZlcjEwggEiMA0GCSqGSIb3DQEB
AQUAA4IBDwAwggEKAoIBAQDpQJo+Izt8v0zriSWwrChc1tnVj3E3h3XuyEHub7hj
y4bMu2PqKByFNr+iikEF3u0d6HrCRSDKt1BcLzL3TsTJ/hJBHAlTyqEgVce1knjL
2g9NnDbekRtJSJCxS9j+RWtP43Xdg5edb5hTCZqdNFHD8oNuSMGFBbHN4oi9eDXl
rvyVHJe+UkI1Ow6mW0+ln/IoKNFPovz+l+ds3fJ5+UHe2TaQPQc7tGZ33j7wfJQd
es8baFdK+lnmGdUOrW9BQE6ReMSezkz6dKdIZdy7jEs6xoflOzyWlgydmnkAvLnx
MBQDgDUbc5MuooVMAWa4yhtz0B9ZmdJDb8jzHDpTPqdRAgMBAAGjgY8wgYwwHQYD
VR0OBBYEFF8KPhl1xxV0110JiBs5iUEPoJ1IMFAGA1UdIwRJMEeAFF8KPhl1xxV0
110JiBs5iUEPoJ1IoRmkFzAVMRMwEQYDVQQDDAp2cG5zZXJ2ZXIxghQPZTF0TCC+
```

```
GGvTOrl8DEgMi54HYTAMBgNVHRMEBTADAQH/MAsGA1UdDwQEAwIBBjANBgkqhkiG
9w0BAQsFAAOCAQEAMnRLz3CBApSrjfUKsWYioNGQGvh77Smh/1hPGIu4eEldQSmZ
Aj7qclEaORdBxmqrVtA3Z9cX1L0xFrg14nLyddmuWHG3ZChc5ZMpYtD2YpOH265B
FFjDp96vK13dpixWKrVpvakLCCA4EvnC8CEjbm0oNFiCgSwKAoJFCcUzwC33swsU
B2w5/iT6CZKuKhSmET1IDpG8krGC/Ib2GNAS0szMI94P0ajZgVznMcXOJ7gUg4rM
sEB8OzM6GBEZTqbAa9uVMZnOZvZA5jGIbBuelUo0bqGdAyx2B68zzuL//qvsHsvw
kZCyKIaXH0NBV7vexMKWcwFLLBzWizFQbbFpFA==
-----END CERTIFICATE-----
</ca>

# vpnclient1.cert
<cert>
-----BEGIN CERTIFICATE-----
MIIDVjCCAj6gAwIBAgIQLhO4FTrqN5WZiQETULAwnzANBgkqhkiG9w0BAQsFADAV
MRMwEQYDVQQDDAp2cG5zZXJ2ZXIxMB4XDTIxMDIyMTE4NTYzM1oXDTI0MDIwNjE4
NTYzM1owFTETMBEGA1UEAwwKdnBuY2xpZW50MTCCASIwDQYJKoZIhvcNAQEBBQAD
ggEPADCCAQoCggEBALUFYXwk6JW/hRtoMs0Ug5jMcWXsjMUsCz8L8CeXNOs3wQrf
YBWF1TYCLPd2/vwXsvbqCE85IZwjsJ5mEx9YgQ5M1teDkLZqBn8y7VIyDAAU8RsN
NcrnpeMDV0LgZIBeUrHi4ZTooaw4FdJ5BBYRHR1APVaaHDWx59ohJuBDpriWhvWk
lWX0rpSJltXriIOCzky/yEwfw6ah5jWaTgfe41fXq8j3lx2IbgIL7I4//jhC6JYz
N7huTdT2uB2MUbYX0XWBffMG8wcBZtMI2XryZmPvFYWP7N5nZZsBXkLz/UngAu3k
jkYJOnJy/hdOFLN/yXj7VFydmivUSeekdjjxyAECAwEAAaOBoTCBnjAJBgNVHRME
AjAAMB0GA1UdDgQWBBSnLIQoTPLyECbJHfgYBHvQpcmfgzBQBgNVHSMESTBHgBRf
Cj4ZdccVdNddCYgbOYlBD6CdSKEZpBcwFTETMBEGA1UEAwwKdnBuc2VydmVyMYIU
D2UxdEwgvhhr0zq5fAxIDIueB2EwEwYDVR0lBAwwCgYIKwYBBQUHAwIwCwYDVR0P
BAQDAgeAMA0GCSqGSIb3DQEBCwUAA4IBAQBaBpYZXVYUzOcXOVSaijmOZAIVBTeJ
meQz9xBQjqDXaRvypWlQ1gQtO8WnK9ruafc1g/h7LtvqtiALnGiJ0NbshkH8C1KE
yen46UCau5B/Xi0gA7FoPildvYdKSn/jI6KySCsplubjnJK9H/6DjAcEuqFLcsaY
5vpKQGP9Vl7H7hEVs4f1aory1T4Ma/bdXEOqgzHmIARLmxYeJm90sUT/n7e7VXfy
fILZ+8D1fMxCbeQRBkg1e8wJfgEbMRY9aGGt1qAs9gkm9RPelGB18v4iCbyebv3X
4hVHmfjcixdbWiABC7yq/gisooQ0robW/92dgemcwO0awHZX+opNBgwr
-----END CERTIFICATE-----
</cert>

# vpnclient1.key
<key>
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIFHDBOBgkqhkiG9w0BBQ0wQTApBgkqhkiG9w0BBQwwHAQInjFvz5a4mY8CAggA
MAwGCCqGSIb3DQIJBQAwFAYIKoZIhvcNAwcECNsxQXxvMpN0BIIEyEZdgFwPnGup
vyhywXR6l6ihvHK2GRczIgH0mFIiwQDgDjZj2YsEnvSA/P3MHplkU/bgv9DJ5j2T
C5wPDmGN4yG1boHx9BQKbXqxGwdz/UcHwmNKur9qnSFrSVEvMDwvum+rmzWuKykf
gkKKBCT1JZ2DWKtjjDNYG9qhBn3S2zYVq311dDuLbBcruvo1UL031sDDYWTpVuuf
zZc0ozng0Nzb35bNkG6Ib+LYLzJi4stxzw0DTFl52lKv++R6xhmqb81IJE3vBs4H
DOutkYfifO1eGqEKksPQRl8n03UVkOtB5pH8VdQeLqEBBaq3qeIfU6FkH9XrPR/E
8VOg9BNpbyuUW7bQu0MzuJ8Ofkjy9K+HHdwFtGPyOatkeaXT/qcKVMvzWcbr8bPc
VncavzXdzo0Sb8FigsKYU1lNjgo00Phd3m0AOfptrweK6ucBds5SmqNrUFXiQ2JA
Ms3LUw4CXBBgvdu5TsA2xLGysip0RPKLyTnUPGnXxbBaaHMv8Jz3XRCrWgZbtAE3
XhE9fKw+ZMEP+2jpC/1mjN/N9VuJfYZEhgA84wzYMu6pt3zPkWZqR6yGTDFEDhvh
OAZYEpqrhe++nxDpuQlpCCl4IndSg9L9oX1ydrvPNHGbRVztd3+r9wr4Ub3fJ1g/
9ckCdanohEymKbjw34HEMmdx+fn5k2T9bLnl8fsYtcESkg04ChON3yOnZFKl6chT
BQ9X2Qmeg7FoawWiUY5o+7OHNKL7QpRt4jXPbXNuXFK9EYvuRzUqubLhL5DdmjuO
Se1vvZg7fT4C8qjYsoCa18idA00EN3ePFFf9AssHCoVW92GiUTTKG+qURCjtNtG6
```

```
dnPvxiSf98OBkkjeX3ni0cKdfMGoQTSdEy5GexvfRMF5HJrGO+CWXmqSBsuIlPUe
quqCsPmpaT2Ws/0UU9cKe4qaKjTL7CghtFmUEhH7t6Cd41Ki9gKi33j3541l9w7l
J1bgca4rRUCecp2BPF3IjJc/RnTvHkbUK4mDX9s8xJhYf9WE6JYsk3NBSNNIj/9G
FMJlo71x8H3OAdFzRN5bjV797HByZ+YidZIgGAx2dSko3PQPy7RSxdmzFbxfUvzj
9jcYEu+V9unbtDK2qZ9I+LqXGE+EXjPBui40IWp8XIYNlSLn2qgroH079lXhXKBY
+DzcBzyT7GTX2QeYE+yqqPRIFWHnbnsnD6dMnAa46h+Si+f5sq33rfRsF7UpK4gV
IhzFkncCM47/Taqi0OY04Q40LuSCDjmjFL+VzZOsAtWGRNYNzIgniThEehElJwfI
ErzClcVptjhtCer8BPuO7YaMIHk1hKecHFqw3RrimWzroL1iu9Q29m2oM+bVc6mD
we6r+t8JbaAFxoHBK4i6M0rcdJPICxDTIOjPC3Fg/MeqiCi7F0DFZvXwPGRD+0Of
MBnsDplEUjK06jbE5BjGQ7n7P+dwDxyp/aVO4CfX7ZOco6h9r3b6nqlzPVNE9erw
kS7WwT/TWraw/sfIO9sNSgle7PoRh2s/w/oGVhC6ymlMdXe+mhMzHFnGEbBRh2Rd
kd/EdYNubHg0k9+RLTwbgwZ+176cIJyOpqaoJGv0bsKM8X26Pk/fkyF6xgdQYQOx
8i9Whea8OjUOQAcgc7gUyA==
-----END ENCRYPTED PRIVATE KEY-----
</key>

# ta.key
<tls-auth>
-----BEGIN OpenVPN Static key V1-----
4eb35b44d1d8a82cfa51af394d4f58f3
69bf8fe8c0a0a032f38b0ee104889628
8a5dc89486736b39d64ad3c6831bf9ba
9f3f96c3307d322a5bf055b9bc3bfa74
929faf361c14de97445f5927794264bb
e3f71c925f2236cfb0109ecfd6406cef
857dfb39783a09ecd56d3cf09ebbc853
0f43b1c787f0db99dbecabcd2090cfbb
54c86d8102a5430fd6a7f37ab5ce8ed9
f6bec8984bde4267f78913ff702dd396
a205b6be9e7ab41cf1ebad3953c27c7c
f3b435345e02aede049ef7c9f1c2704f
2ed91110ccb19d0d3bd46a00f54c73e2
07b31160cdc54c3f5a7989bb999ac5f3
89c6de7e79fc93399924a8d298eab462
231234e690c319d5cbd832788f0dbcfb
-----END OpenVPN Static key V1-----
</tls-auth>
```

Now you have only one file to distribute to your clients. (See Recipe 13.1 to learn what to install for Linux, macOS, Windows, iOS, and Android clients.)

The easy way to import a new *.ovpn* file in Linux is using NetworkManager. Open "VPN Connections" → "Add a VPN connection." This opens "Choose a VPN Connection Type." Select "Import a Saved VPN Connection," click Create, and find your *.ovpn* file in the file selector. Review the settings on the General and VPN tabs.

On the General tab, make sure that "All users may connect to this network" is not checked. This is a simple but important security measure that requires every user to have their own individual OpenVPN configuration.

On the VPN tab, note that NetworkManager converts the inline certificates to *.pem* files (Figure 13-2). This is normal and not a mistake. You can compare these to the originals; click on the little file folder icon to the right to see where the converted files are stored.

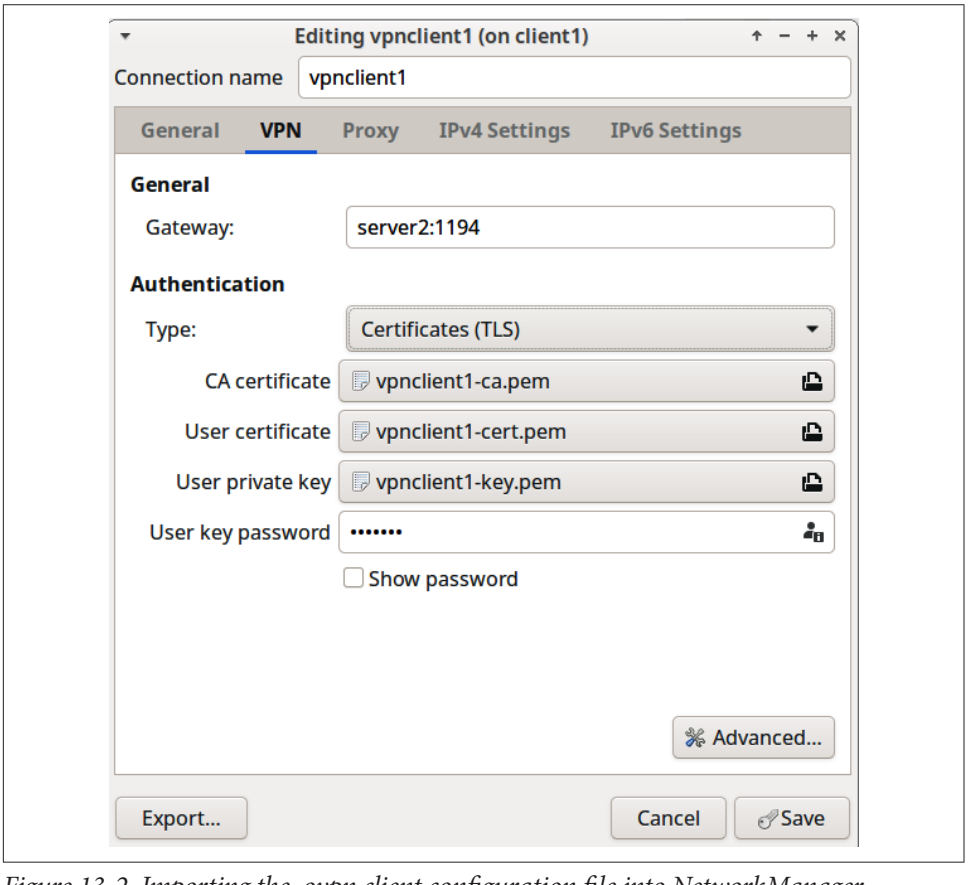


Figure 13-2. Importing the .ovpn client configuration file into NetworkManager

For all other clients, the procedure is similar. Follow their instructions, and if all goes well, your clients will be up and running in a couple of minutes.

The NetworkManager import function also works with client configuration files that are not inline, like in Recipe 13.7. In this case the certificates are not converted, and they retain their original filenames.

An inline file may have the *.conf* extension if you have only Linux clients.

See Also

- EasyRSA
- OpenVPN documentation
- *man 8 openvpn*
- OpenSSL Cookbook

13.10 Hardening Your OpenVPN Server

Problem

You want to know some options for making your OpenVPN more secure.

Solution

The OpenVPN default settings are pretty good, but they're designed for broader compatibility. There are some changes you can make that will make your server stronger.

The following examples go in both server and client configuration files. These options maximize the effectiveness of TLS. All SSL and TLS protocols older than TLS 1.2 are deprecated and should not be allowed. Accept only TLS 1.2 or higher:

```
tls-version-min 1.2
tls-version-max 1.3 or-highest
```

Use a stronger data channel cipher, and enforce its use by disabling cipher negotiation:

```
AES-128-GCM
ncp-disable
```

There are many changes in TLS 1.3, so you need two different configurations for TLS 1.2 and 1.3. These are all stronger and more efficient encryption ciphers:

```
# TLS 1.3
tls-ciphersuites TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256
# TLS 1.2
tls-cipher TLS-ECDHE-ECDSA-WITH-CHACHA20-POLY1305-SHA256:TLS-ECDHE-RSA-
WITH-CHACHA20-POLY1305-SHA256:TLS-ECDHE-ECDSA-WITH-AES-128-GCM-SHA256:
TLS-ECDHE-RSA-WITH-AES-128-GCM-SHA256
```

Use Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) in place of our old Diffie-Hellman static keys. You do not have to create a *ta.key*, as you do in Recipe 13.5:

```
dh none
ecdh-curve secp384r1
# use tls-server on the server, tls-client on the client
tls-server
```

Add the *float* option, in the server configuration only, to allow clients to roam on different networks without losing connection, as long they pass all other authentication tests.

The *opt-verify* option, in the server configuration only, checks for compatibility between server and client settings, and disconnects clients that do not match. *opt-verify* checks *dev-type, link-mtu, tun-mtu, proto, ifconfig, comp-lzo, fragment, keydir, cipher, auth, keysize, secret, no-replay, no-iv, tls-auth, key-method, tls-server*, and *tls-client*.

See the Discussion for complete example configurations.

Discussion

Put these enhancements all together in the server configuration:

```
# vpnserver1.conf
port 1194
proto udp
dev tun
user nobody
group nobody

ca /etc/openvpn/keys/ca.crt
cert /etc/openvpn/keys/vpnserver1.crt
key /etc/openvpn/keys/vpnserver1.key

server 10.10.0.0 255.255.255.0
ifconfig-pool-persist ipp.txt
keepalive 10 120
persist-key
persist-tun
tls-server

remote-cert-tls client
verify-client-cert require
tls-cert-profile preferred
tls-version-min 1.2
tls-version-max 1.3 or-highest

float
opt-verify
AES-128-GCM
ncp-disable
dh none
ecdh-curve secp384r1

# TLS 1.3
tls-ciphersuites TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256
# TLS 1.2
tls-cipher TLS-ECDHE-ECDSA-WITH-CHACHA20-POLY1305-SHA256:TLS-ECDHE-RSA-
```

```
WITH-CHACHA20-POLY1305-SHA256:TLS-ECDHE-ECDSA-WITH-AES-128-GCM-SHA256:
TLS-ECDHE-RSA-WITH-AES-128-GCM-SHA256

status openvpn-status.log
verb 4
mute 20
explicit-exit-notify 1
```

An example client configuration, using the inline file format (Recipe 13.9):

```
# vpnclient1.conf
client
dev tun
proto udp
remote server1 1194

persist-key
persist-tun
resolv-retry infinite
nobind

user nobody
group nobody
tls-client
remote-cert-tls server
verb 4

# Using inline keys
# ca.crt
<ca>
[...]
</ca>

# client.crt
<cert>
[...]
</cert>

# client.key
<key>
[...]
</key>

tls-version-min 1.2
tls-version-max 1.3 or-highest
AES-128-GCM
ncp-disable
dh none
ecdh-curve secp384r1

# TLS 1.3 encryption settings
tls-ciphersuites TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256
# TLS 1.2 encryption settings
```

```
tls-cipher TLS-ECDHE-ECDSA-WITH-CHACHA20-POLY1305-SHA256:TLS-ECDHE-RSA-WITH-
CHACHA20-POLY1305-SHA256:TLS-ECDHE-ECDSA-WITH-AES-128-GCM-SHA256:TLS-ECDHE-RSA-
WITH-AES-128-GCM-SHA256

status openvpn-status.log
verb 4
mute 20
explicit-exit-notify 1
```

These options strengthen the authentication between client and server, and enforce using TLS 1.2 and higher. If you're wondering how to know which ciphers and ciphersuites to use, I asked some experts. You can lock down your setup even tighter. For example, disallowing users from saving passwords, adding more restrictions to client-server authentication, using SELinux, or using a chroot. These are advanced topics not covered here.

See Also

- EasyRSA
- OpenVPN documentation
- *man 8 openvpn*
- OpenSSL Cookbook

13.11 Configuring Networking

Problem

Your OpenVPN server is running, all of your connection tests work as expected, and now you need to know how to set up your networking so that remote clients can find your server and traffic gets routed appropriately.

Solution

There is no one-size-fits-all solution. Configuring your networking has to take into account how your LAN is set up, whether you are connecting individual clients or linking networks, IPv4, IPv6, how your internet gateway is set up, and lots more. Consult Jan Just Keijser's excellent *OpenVPN Cookbook*, 2nd Edition (O'Reilly), to find the answers you need. This book covers TUN versus TAP, Windows clients, PAM and LDAP, IPv6, routing, and site-to-site configurations.

Discussion

Networking is the most challenging part of running any server, especially an important security server. It pays to study this and take pains to get it right. The OpenVPN documentation has a lot of good information on networking as well.

See Also

- OpenVPN documentation
- *man 8 openvpn*

CHAPTER 14

Building a Linux Firewall with firewalld

This chapter covers the basics of using firewalld to build host firewalls. Individual hosts have different requirements. For example, a server has to allow different types of incoming connection requests, and a PC running no services does not have to accept any connection requests. A laptop that is used to access multiple networks needs dynamic firewall management.

firewalld Overview

firewalld, like all firewalls, has a very long list of capabilities. We will mainly learn about using firewalld *zones* to control traffic entering our systems. A zone is a container for a level of trust; for example, some zones allow all manner of incoming connection requests, and some are very restrictive. Each network interface on a system may be assigned only one zone, and one zone may be assigned to multiple interfaces.



Networking Knowledge Required

The most important networking concepts to understand are ports, services, TCP, UDP, port forwarding, masquerade, routing, and IP addressing. You will understand how to configure your firewall when you understand these. If you need some coaching on computer networking, try *Networking Fundamentals* by Gordon Davies (Packt Publishing), or *Networking All-in-One For Dummies*, 7th Edition by Doug Lowe (For Dummies). If you have an O'Reilly Learning Platform subscription, you will find a wealth of great information.

The traditional Linux firewall is built with the *netfilter* packet-filtering framework in the Linux kernel, which filters incoming and outgoing network traffic, and *iptables*, which is the software used to create and manage tables of rules to filter your traffic.

Times change, and iptables is being replaced by newer rules managers, such as *ufw* (Uncomplicated Firewall), *nftables* (Netfilter tables), and *firewalld* (firewall daemon). firewalld, like iptables and nftables, uses tables of rules to manage traffic filtering. It provides both a command-line interface and a nice graphical interface, *firewall-config*. firewalld is a frontend to both iptables and nftables. nftables is a significant improvement over iptables and is intended to be the default backend for firewalld, but in some Linux distributions iptables is still the default. Set your preferred backend in */etc/firewalld/firewalld.conf* with the *FirewallBackend* option (Recipe 14.4).

firewalld comes with predefined sets of rules, called *zones*, for different use cases, such as a machine running no services, a machine running services, and different zones for different network interfaces on the same machine. You can edit these zones to suit your own requirements.

firewalld zones manage *services*, which are configurations for common services such as ssh, imaps, and rsync. Most of the predefined services include only the standard port assignments. You may edit these as you need and create your own custom zones.

firewalld is integrated with NetworkManager, so you don't have to worry about managing dynamic connections, like when you tote your laptop all over and connect to different networks.



The NetworkManager Service

NetworkManager has been an important part of Linux since 2004. NetworkManager replaced a hodgepodge of cumbersome network client tools and manages all of your network interfaces and network connections. If you're not familiar with NetworkManager, see GNOME NetworkManager.

If you are running public servers on a commercial hosting service, your firewall setup depends on what your service provider supports. Protecting public servers, such as web servers and online storefronts, whether they are remotely hosted or in your own datacenter, requires a great deal of skill and care that is beyond the scope of this book. Do please get in-depth study and training, or hire experts.

How Firewalls Work

Once upon a time, Ubuntu Linux did not ship with a firewall, because the default installation had no public services, and therefore no listening network ports. The reasoning was that with no listening ports there were no points of attack. Fortunately,

this decision was reversed in later releases, because users make changes, even the most experts users make mistakes, and attackers are always discovering new vulnerabilities. Security is a multilayered process.

Let's look at how firewalls work. The basic principle is deny all, allow only as needed.

A network service, such as an SSH server, needs to open a network port to enable remote users to log in. You are allowing other people into your system. The default port for *sshd* is TCP port 22. You can see all the listening ports on your system with the *netstat* command. This snippet shows what the SSH port looks like:

```
$ sudo netstat -untap | sed '2p;/ssh/!d'
Proto Recv-Q Send-Q Local Address  Foreign Address  State   PID/Program name
tcp        0      0 0.0.0.0:22     0.0.0.0:*        LISTEN  1296/sshd: /usr/sbi
tcp6       0      0 :::22          :::*             LISTEN  1296/sshd: /usr/sbi
```

This example shows that there is not an active connection because the Foreign Address fields are all zeroes and the State is LISTEN. *sshd* is listening for incoming IPv4 and IPv6 connections on all network interfaces and all IP addresses on TCP port 22. The combination of IP address and port number is an address that tells the Linux kernel where to send SSH packets.

This example shows an active SSH connection, with the ESTABLISHED State. It lists the local address and port that the remote machine is connected to, and the foreign address and port of the remote machine (the Recv-Q and Send-Q columns have been removed for clarity):

```
$ sudo netstat -untap | sed '2p;/ssh/!d'
Proto  Local Address     Foreign Address      State       PID/Program name
tcp    0.0.0.0:22        0.0.0.0:*            LISTEN      1296/sshd: /usr/sbi
tcp    192.168.1.97:22   192.168.1.91:56142   ESTABLISHED 13784/sshd: duchess
tcp6   :::22             :::*                 LISTEN      1296/sshd: /usr/sbi
```

There are several ways to control which TCP/IP packets can access a particular IP address and port. Most servers have configuration options to listen only on particular network interfaces or IP addresses, and to accept requests from specific addresses and address ranges. A firewall adds additional controls, and it is a best practice to use both.

Network Ports and Numbering

There are 65,536 possible network ports on a Linux system, numbered 0-65535, and many of them are reserved for specific services. 0 is reserved and not used. You can see all of these in the */etc/services* file, which is on every Linux. See the IANA Service Name and Transport Protocol Port Number Registry for the complete official list.

This is how the port numbering ranges are organized:

- 0-1023 are called the *well-known ports*. These are system ports for common services, such as FTPS (secure file sharing), SSH (secure remote login), NTP (Network Time Protocol), POP3 (email), HTTPS (encrypted web server), and so on.
- 1024-49151 are the *registered ports*, which are for additional services.
- 49152-65535 are the *ephemeral ports*, also called private ports and dynamic ports. These are used by your system to complete connections with remote services. For example, when you are web surfing, it looks like this in *netstat* (the Recv-Q and Send-Q columns have been removed for clarity):

```
$ sudo netstat -untap
Proto  Local Address         Foreign Address    State        PID/Program name
[...]
tcp    192.168.43.234:50586  72.21.91.66:443    ESTABLISHED  2798/firefox
tcp    192.168.43.234:38262  52.36.174.147:443  ESTABLISHED  6481/chrome
tcp    192.168.43.234:53232  99.86.33.45:443    ESTABLISHED  2798/firefox
[...]
```

This illustrates a response to an outgoing request from your computer. When you visit a website you initiate the connection request, and the remote web server sends responses to ephemeral network ports on your system. The first connection on the list is connected to the example local computer at IP address 192.168.43.234, port 50586. The Foreign Address is the remote server's IP address and port. The state ESTABLISHED means it is connected to another machine. When the session is finished, after closing the web browser, port 50586 is released and ready to be used again.

Ephemeral ports are not listening ports for services. Connections to ephemeral ports are temporary, and are created only as replies to an outgoing connection request from your computer, such as visiting a website. A firewall can block ephemeral ports, but then you have no access to hosts or sites outside of your computer.

14.1 Querying Which Firewall Is Running

Problem

You need to know which firewall your Linux system is using.

Solution

Start with the documentation for your particular Linux distribution, as most Linuxes install with a firewall. The three most common are *iptables* (Internet Protocol tables), *ufw* (Uncomplicated Firewall), and *nftables* (Netfilter tables). All three manage filter rules on the netfilter framework, which is part of the Linux kernel.

Then see what systemd says. This example shows that nftables is running:

```
$ systemctl status nftables.service
    ● nftables.service - Netfilter Tables
     Loaded: loaded (/usr/lib/systemd/system/nftables.service; disabled; vendor>
     Active: active (exited) since Sat 2020-10-17 13:15:05 PDT; 4s ago
       Docs: man:nft(8)
    Process: 3276 ExecStart=/sbin/nft -f /etc/sysconfig/nftables.conf (code=exi>
   Main PID: 3276 (code=exited, status=0/SUCCESS)
   [...]
```

This shows firewalld is running:

```
$ systemctl status firewalld.service
● firewalld.service - firewalld - dynamic firewall daemon
Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor>
     Active: active (running) since Sat 2020-10-17 12:36:20 PDT; 37min ago
       Docs: man:firewalld(1)
   Main PID: 775 (firewalld)
      Tasks: 2 (limit: 4665)
     Memory: 40.9M
     [...]
```

This example checks for ufw and shows that it is installed but inactive:

```
$ systemctl status ufw.service
● ufw.service - Uncomplicated firewall
     Loaded: loaded (/lib/systemd/system/ufw.service; disabled; vendor preset:
enabled)
     Active: inactive (dead)
       Docs: man:ufw(8)
```

If any of them are not installed, you will see a message to that effect.

You could remove ufw and nftables, or mask them so that they cannot be started:

```
$ sudo systemctl stop ufw.service
$ sudo systemctl mask ufw.service

$ sudo systemctl stop nftables.service
$ sudo systemctl mask nftables.service
```

Discussion

It is best to run only one firewall, unless you enjoy untangling conflicting firewall rules.

See Also

- Chapter 4
- *https://firewalld.org*

14.2 Installing firewalld

Problem

You need to install firewalld on your Linux system.

Solution

If firewalld is not on your system, install the *firewalld* package, and install *firewall-config* to get the nice graphical interface.

Discussion

So far, the major Linux distributions, thankfully, all use the same package names, *firewalld* and *firewall-config*.

firewalld may or may not start automatically after installation, depending on your Linux distribution. It must be running to create and test rules.

If possible, disable your machine's network connection until you have completed your initial firewalld configuration. Disconnect from your network by clicking the NetworkManager applet, which is installed by default on most Linux distributions (Figure 14-1).

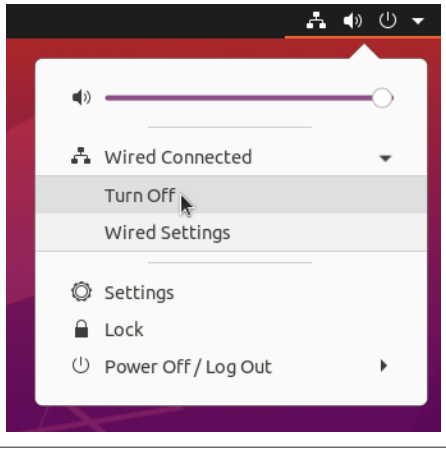


Figure 14-1. Disconnect from the network with NetworkManager

Or use the *nmcli* command. The following example finds and disconnects a WiFi connection. Use the CONNECTION name in your command:

```
$ nmcli device status
DEVICE  TYPE  STATE        CONNECTION
wlan0   wifi  connected    ACCESS_POINTE
```

```
$ nmcli connection down ACCESS_POINTE
Connection 'ACCESS_POINTE' successfully deactivated
(D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection/4)
```

Restore the connection:

```
$ nmcli connection up ACCESS_POINTE
Connection successfully activated
(D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection/7)
```

Manage firewalld the usual way with systemd. Here are the commands:

- *systemctl status firewalld.service*
- *sudo systemctl enable firewalld.service*
- *sudo systemctl start firewalld.service*
- *sudo systemctl stop firewalld.service*
- *sudo systemctl restart firewalld.service*

See Also

- Chapter 4
- *https://firewalld.org*
- The Appendix

14.3 Finding Your firewalld Version

Problem

You need the version number of your installed firewalld.

Solution

Query the installed package with your package manager, or use *firewall-cmd*:

```
$ sudo firewall-cmd --version
0.9.3
```

Discussion

firewalld must be running for the *firewall-cmd* command to work. If it is not running you will see a "FirewallD is not running" message.

See Also

- *https://firewalld.org*

14.4 Configuring iptables or nftables as the firewalld Backend

Problem

You want to choose your own firewalld backend, either iptables or nftables.

Solution

Edit */etc/firewalld/firewalld.conf* with your preference:

```
FirewallBackend=nftables
```

Or:

```
FirewallBackend=iptables
```

Then restart firewalld.

Discussion

You may need to install your preferred backend.

Even if you don't care which one your system uses, you should use nftables because that is what the firewalld developers are actively working on.

See Also

- "firewalld Overview" on page 325
- The nftables backend blog post by the firewalld developers has detailed information about the two backends and future development.

14.5 Listing All Zones and All Services Managed by Each Zone

Problem

You want to see all the available zones in your firewalld configuration and the services that each zone manages.

Solution

List the default zone:

```
$ firewall-cmd --get-default-zone
public
```

List all zones:

```
$ firewall-cmd --get-zones
block dmz drop external home internal public trusted work
```

List all active zones, the zones currently in use:

```
$ firewall-cmd --get-active-zones
internal
  interfaces: eth1
work
  interfaces: wlan0
```

List the configuration of a zone:

```
$ sudo firewall-cmd --zone=public --list-all
public
  target: default
  icmp-block-inversion: no
  interfaces:
  sources:
  services: dhcpv6-client ipp ipp-client mdns ssh
  ports:
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

List the configurations of all zones:

```
$ sudo firewall-cmd --list-all-zones
[...]
```

Discussion

firewalld zones define levels of trust for network connections. Each zone contains the zone description and other items as shown in the preceding example for the *public* zone. Zone files are in XML format, and must have the *.xml* file extension. Look in */usr/lib/firewalld/zones* to see their source files.

The following list defines zone options:

- *target:* defines the default action for packets that do not match any rules. It takes one of four values: *default*, *ACCEPT*, *DROP*, or *REJECT*. For example, when connection requests for dhcpv6-client, ipp, ipp-client, mdns, or ssh packets arrive in the example *public* zone, they are accepted. Any packets that do not match the allowed services are rejected by the *default* target, and it sends reject messages.
  — *ACCEPT* accepts all packets that are not explicitly blocked by rules.
  — *DROP* silently drops all packets not explicitly allowed.
  — *REJECT* is similar to *DROP*, except that it also sends reject messages.
- *icmp-block-inversion* inverts your ICMP requests settings. Any requests that are blocked are changed to unblocked, and unblocked requests are inverted to blocked. It is usually set to *no*.
- *interfaces:* defines the network interface or interfaces that this zone is applied to. Each interface may be bound to only one zone, and you may use the same zone on multiple interfaces.
- *source:* takes IP and MAC addresses, and IP address ranges. For example, you can accept only packets from your local network, from specific hosts, or block hosts or networks.
- *services:* is the list of services managed by this zone.
- *ports:* list port numbers managed by this zone.
- *protocols:* list additional TCP protocols managed by this zone, as shown in */etc/protocols*.
- *masquerade:* is either *yes* or *no*. Masquerading is for sharing an IPv4 internet connection. Set it to *no* on all hosts except routers.
- *forward-ports:* is for forwarding packets that come in on one port to another port.
- *source-ports:* is for listing source ports.
- *icmp-blocks:* is for listing ICMP types to block.
- *rich rules* are custom rules that you write.

See Also

- *https://firewalld.org*
- *man 5 firewalld.zone*
- *man 1 firewall-cmd*

14.6 Listing and Querying Services

Problem

You want to see a list of services that firewalld supports.

Solution

Use the *firewall-cmd* command:

```
$ sudo firewall-cmd --get-services
RH-Satellite-6 amanda-client amanda-k5-client amqp amqps apcupsd audit bacula
bacula-client bb bgp bitcoin bitcoin-rpc bitcoin-testnet bitcoin-testnet-rpc
bittorrent-lsd ceph ceph-mon cfengine cockpit condor-collector ctdb dhcp dhcpv6
[...]
```

That is rather a big glob. Convert it to nice tidy single column:

```
$ sudo firewall-cmd --get-services| xargs -n1
RH-Satellite-6
amanda-client
amanda-k5-client
amqp
amqps
apcupsd
[...]
```

Create more columns with *xargs -n2*, *xargs -n3*, and so on.

firewalld services can be more than simple port addressing. For example, the *bittorrent-lsd* service includes two destination IP addresses:

```
$ sudo firewall-cmd --info-service bittorrent-lsd
bittorrent-lsd
  ports: 6771/udp
  protocols:
  source-ports:
  modules:
  destination: ipv4:239.192.152.143 ipv6:ff15::efc0:988f
  includes:
  helpers:
```

The *ceph-mon* service opens two listening ports:

```
$ sudo firewall-cmd --info-service ceph-mon
ceph-mon
  ports: 3300/tcp 6789/tcp
  [...]
```

You may edit any of the predefined services to meet your requirements.

Discussion

When you're adding services to a zone, use their names exactly as they appear in the list. You can create your own custom service; see "Add a Service" in the firewalld documentation.

See Also

- *https://firewalld.org*
- "Add a Service" in the firewalld documentation

14.7 Selecting and Setting Zones

Problem

You want to know how to select and set the right zone.

Solution

The firewalld zone you select depends on which services your machine is running. If your machine is not running any network services and needs only a network connection, use the *drop* or the *block* zone. The *drop* zone is the most restrictive, dropping all incoming connections requests, and allowing replies only to connections initiated from the computer. *block* is like *drop*, except it sends rejection messages.

The other zones are configured differently on the different Linux distros, so you need to see how they are configured on your system, like this example for the *work* zone:

```
$ sudo firewall-cmd --zone=work --list-all
work
  target: default
  icmp-block-inversion: no
  interfaces:
  sources:
  services: dhcpv6-client ssh
  ports:
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

You must bind a zone to a network interface. The following example assigns the *work* zone to eth0, then verifies it:

```
$ sudo firewall-cmd --zone=work --permanent --change-interface=eth0
success
```

```
$ sudo firewall-cmd --zone=work --list-interfaces
eth0
```

If you prefer to test changes before making them permanent, omit the *--permanent* option. This creates a *runtime* configuration, and the change is immediately applied. Runtime changes are lost when firewalld is restarted and when you run *firewall-cmd --reload*. Convert runtime changes to permanent:

```
$ sudo firewall-cmd --runtime-to-permanent
```

It is not necessary to reload the firewalld configuration when you bind a zone to a network interface or restart firewalld.

Discussion

How do you know which zone to select? These are the predefined zones that come with firewalld on Ubuntu 20.04, in order from most restrictive to least restrictive. Zones may be configured a little differently on your Linux; see Recipe 14.5 to learn how view your zone configurations.

The following list describes the default zones:

drop
: All unsolicited incoming network packets are dropped, and there is no reply. Only incoming packets that are replies to connections initiated from your computer are allowed. This is the strongest protection when you are connected to an untrusted network and don't need to allow access for incoming SSH connections, shared files, or any other external connection requests.

block
: Any incoming network connections are rejected with an *icmp-host-prohibited* message for IPv4, and *icmp6-adm-prohibited* for IPv6. Only network connections initiated from your system are allowed.

public
: Incoming dhcpv6-client, ipp, ipp-client, mdns, and ssh connections are accepted, all others are blocked.

external
: This is for a simple internet gateway, combining a firewall and simple routing. Only incoming SSH connections are accepted, and IPv4 masqerading is enabled for sharing an internet connection.

dmz
: For computers in your demilitarized zone that are publicly accessible. Only incoming SSH connections are accepted. (A DMZ is a separate network segment on your network for internet-facing servers.)

work
: Only incoming ssh and dhcpv6-client connections are accepted.

home
: Only incoming ssh, mdns samba-client, and dhcpv6-client connection requests are accepted.

internal
: Only incoming ssh, mdns, samba-client, and dhcpv6-client connection requests are accepted.

trusted
: All network connection requests are accepted.

You may customize any of these zones or create new zones; see Recipe 14.9.

See Also

- Recipe 14.9
- *https://firewalld.org*

14.8 Changing the Default firewalld Zone

Problem

You don't like your default firewalld zone and want to change it.

Solution

Verify your current default:

```
$ firewall-cmd --get-default-zone
internal
```

Suppose you want *drop* as your default, because it is the most restrictive. Set the new default with the *firewall-cmd* command:

```
$ sudo firewall-cmd --set-default-zone drop
success
```

It is not necessary to reload the firewalld configuration or restart firewalld when you use this command.

Discussion

You may assign zones with NetworkManager (Recipe 14.11). NetworkManager assigns the default zone to all connections that you do not explicitly assign a zone to.

See Also

- The Discussion in Recipe 14.7 to learn about firewalld zones
- Recipe 14.11
- *https://firewalld.org*

14.9 Customizing firewalld Zones

Problem

None of the default zones meet your needs, and you want to modify the predefined zones.

Solution

Suppose you like the *internal* zone, but the default configuration isn't quite what you want. The current configuration allows *ssh*, *mdns*, *samba-client*, and *dhcpv6-client*:

```
$ firewall-cmd --zone=internal --list-all
internal
  target: default
  icmp-block-inversion: no
  interfaces:
  sources:
  services: ssh mdns samba-client dhcpv6-client
[...]
```

The following example shows how to remove *samba-client* because you don't use Samba:

```
$ sudo firewall-cmd --remove-service=samba-client --zone=internal
success
```

You are running a small local 389 Directory server, so you need to add the LDAPS service:

```
$ sudo firewall-cmd --zone=internal --add-service=ldaps
success
```

These are temporary changes that will not survive a reboot or configuration reload. However, they are immediately applied so that you can test them. Test your changes, and if everything works as expected, make the changes permanent:

```
$ sudo firewall-cmd --runtime-to-permanent
success
```

To discard the changes, do not use *--runtime-to-permanent*. Instead, use *--reload* to discard the runtime changes and revert to your original configuration:

```
$ sudo firewall-cmd --reload
success
```

Discussion

--reload does not interrupt any active connections.

--complete-reload reloads firewalld completely, including reloading kernel modules, and terminates active connections. This is a good option when your runtime changes are so messed up you want to start over.

See Also

- The Discussion in Recipe 14.7 to learn about firewalld zones
- *https://firewalld.org*
- Chapter 4

14.10 Creating a New Zone

Problem

You want to create a new custom zone.

Solution

Create an XML containing your zone configuration, then reload firewalld and it is ready to use.

The following example creates a zone for local name services, with DNS and DHCP servers on the same machine, and SSH access. The example file is named */etc/firewalld/zones/names.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<zone>
  <short>Name Services</short>
  <description>
    DNS and DHCP servers for the local network, IPv4 only.
  </description>
  <service name="dns"/>
  <service name="dhcp"/>
  <service name="ssh"/>
</zone>
```

Run the *sudo firewall-cmd --get-zones* command, and your new zone will not be listed. Add the *--permanent* option to see any new zones that are not yet read by firewalld, and now the new "names" zone appears. Zone names are the filenames without the *.xml* extension:

```
$ sudo firewall-cmd --permanent --get-zones
block dmz drop external home internal names public trusted work
```

Reload firewalld:

```
$ sudo firewall-cmd --reload
success
```

Now firewalld can read it, and you can see it with the other zones:

```
$ sudo firewall-cmd --get-zones
block dmz drop external home internal names public trusted work
```

And list its configuration:

```
$ sudo firewall-cmd --zone=names --list-all
names
  target: default
  icmp-block-inversion: no
  interfaces:
  sources:
  services: dhcp dns ssh
  ports:
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

Your new zone is ready to use, and you can modify it just like any other zone.

Discussion

See *man 5 firewalld.zone* to learn about configuration options and see the source files for the predefined zones in */usr/lib/firewalld/zones/* to use as examples. The only files that go in */etc/firewalld/zones/* are user custom files.

Remove a zone by deleting its *.xml* file, and then reload firewalld.

See Also

- *man 5 firewalld.zone*
- *https://firewalld.org*
- Recipe 14.9

14.11 Integrating NetworkManager and firewalld

Problem

You travel between multiple networks, such as multiple work locations, coffee shops, hotels, and coworking locations. You need to know how to set up NetworkManager to keep up with these changes and always ensure that new connections are assigned to the correct firewall zone.

Solution

NetworkManager includes firewalld integration. When you connect to a new network, NetworkManager assigns it to your default firewalld zone.

You may assign a nondefault zone to a particular connection with NetworkManager. If you have the NetworkManager applet in your panel, click it to bring up the Edit Connections dialog (Figure 14-2).

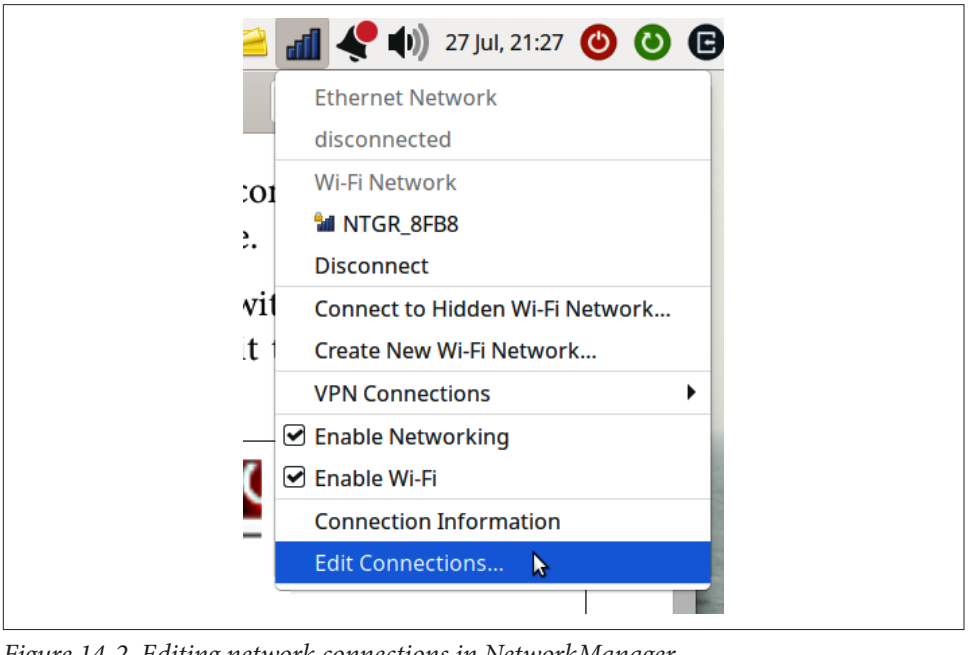


Figure 14-2. Editing network connections in NetworkManager

Or, run the *nm-connection-editor* command to open the editor. Click Edit Connections, click the connection you want to edit, and click the gear icon to open the editor. This opens the editing dialog (Figure 14-3).

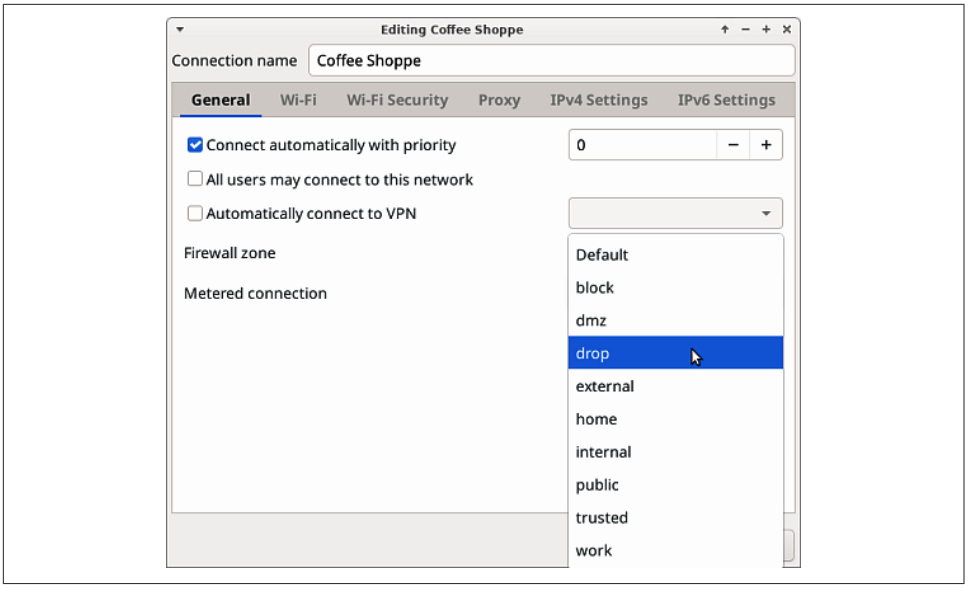


Figure 14-3. Changing the firewall zone

Go to the General tab and use the Firewall Zone drop-down menu to select the zone you want for that connection. Save your change and you're finished.

See Also

- The Discussion in Recipe 14.7 to learn about firewalld zones
- Recipe 14.9
- NetworkManager Reference Manual

14.12 Allowing or Blocking Specific Ports

Problem

You are using nonstandard ports, for example, port 2022 for your SSH server. You want to block port 22 and allow port 2022.

Solution

Any port that is not specifically allowed is denied for all firewalld zones, except the *trusted* zone, which allows everything. If you were using the default SSH service, which uses TCP port 22, first remove port 22 from the relevant zone, then add port

2022, then reload firewalld. In this example the nonstandard port is assigned to the *work* zone:

```
$ sudo firewall-cmd --zone=work --remove-port=22/tcp
success
$ sudo firewall-cmd --zone=work --add-port=2022/tcp
success
```

Verify by listing the zone configuration:

```
$ sudo firewall-cmd --list-all --zone=work
work
  target: default
  icmp-block-inversion: no
  interfaces:
  sources:
  services: ssh
  ports:2022/tcp
[...]
```

When you are satisfied, make your changes permanent:

```
$ sudo firewall-cmd --runtime-to-permanent
```

Discussion

If you see a message like "Warning: NOT_ENABLED: 22:tcp" when you try to remove that port, that means it was not enabled for that zone, and you can go ahead and add your new port.

When you use a nonstandard port, clients connecting to the service must specify that port number. For example, for SSH:

```
$ ssh -p 2022 server1
```

How do you know what ports to use? Every service has its own default ports, which you will find in the service's documentation, and in the */etc/services* file. You may use nonstandard ports, and they must be unused ports between 1024 and 49151. Record your changes in */etc/services*. You also need to set your nonstandard ports in the configuration for your server. See Recipe 12.3 for an example.

See Also

- *https://firewalld.org*
- Recipe 12.3

14.13 Blocking IP Addresses with Rich Rules

Problem

You want to block certain IP addresses.

Solution

Create a *rich rule*, which defines the address to be blocked, and the target, which in the example is *reject*. The following example blocks a single address, and is added to the internal zone:

```
$ sudo firewall-cmd --zone=internal \
 --add-rich-rule='rule family="ipv4" source address=192.168.1.91 reject'
success
```

Test it by pinging from the blocked host. The blocked host should see the "Destination Port Unreachable" message.

If you do not want to keep the rule, run *sudo firewall-cmd --reload* to delete it.

To make it permanent, use the *--runtime-to-permanent* option:

```
$ sudo firewall-cmd --runtime-to-permanent
```

List the rich rules in a zone:

```
$ sudo firewall-cmd --zone=internal --list-rich-rules
rule family='ipv4' source address='192.168.1.91' reject
```

To delete a permanent rich rule, use the *--remove-rich-rule* option:

```
$ sudo firewall-cmd --zone=internal \
        --remove-rich-rule="rule family='ipv4' \
        source address='192.168.1.91' reject"
success
```

You don't have to completely block the offending host, but you can apply blocks to specific services. The following example blocks the source address only for the SSH service:

```
$ sudo firewall-cmd --zone=internal --add-rich-rule='rule family="ipv4" \
 source address=192.168.1.91 service name="ssh" protocol=tcp reject'
success
```

Discussion

You may create multiple rich rules in a zone, though be careful to avoid conflicts.

Once upon a time, a person I had the dubious pleasure of working with thought it was funny to practice penetration testing on his coworkers. Our team ran a number

of test servers on our workstations and made them available to the team. Our wannabe pen-tester was so annoying, we all blocked him at our firewalls.

See Also

- The Discussion in Recipe 14.7 to learn about firewalld zones and options
- *https://firewalld.org*
- *man 5 firewalld.richlanguage*

14.14 Changing a Zone Default Target

Problem

You want to change the default target for a zone.

Solution

List the current target:

```
$ sudo firewall-cmd --zone=internal --list-all
internal
  target: ACCEPT
[...]
```

Change it from *ACCEPT* to *REJECT*, then reload and verify:

```
$ sudo firewall-cmd --permanent --zone=internal --set-target=REJECT
success

$ sudo firewall-cmd --reload

$ firewall-cmd --zone=names --list-all
names
  target: %%REJECT%%
[...]
```

Discussion

The zone target defines the default action for packets that do not match any rules. It takes one of four values: *default*, *ACCEPT*, *DROP*, or *REJECT*.

See Also

- *https://firewalld.org*
- The Discussion in Recipe 14.5 to learn about firewalld zones
- Recipe 14.11

CHAPTER 15

Printing on Linux

Linux relies on CUPS, the Common Unix Printing System, to manage printers. In this chapter you will learn how to install and manage printers, and how to share them over your network. You will learn about the *driverless* future of Linux printing, in which printers will be available to client devices without having to install drivers.

Overview

The key to happy printing on Linux is to select good-quality printers and multifunction devices (printer, scanner, copier, fax) that are well supported on Linux. Which, thankfully, is a lot easier than in olden times. When you select a well supported device, the drivers are included in CUPS, and you do not have to hassle with hunting down and downloading manufacturer drivers.

The next-best option is buying machines that have manufacturer-supplied Linux drivers. This is not my favorite option, as the drivers are often old and not maintained, and you must manually install them. This is more common with multifunction devices (MFDs). For one example, the Brother MFC-J5945DW is my personal machine, and it does not have native CUPS drivers, though it does support driverless printing. It was a good deal and inks are cheap, though in hindsight I really should have bought a machine with native Linux support. Native support is more reliable because once a device is supported in CUPS it is always supported, and you won't be at the mercy of manufacturers who don't maintain their drivers or who discontinue their driver downloads.

The least favorable option is to buy without doing your homework and hope it works. In this case you might be able to use macOS drivers (PPD files) for printers not supported in Linux, though it can take some work if the Macintosh PPD has macOC-specific entries, such as calling macOS executables, libraries, or filters. These must be

replaced with the equivalents for Linux, if they exist. See cupsFilter for some useful information if you want to try this.

If you want a shared device, the least troublesome approach is to get one that has networking built in, and that has built-in controls for copying, setting up networking, viewing ink levels, cleaning print heads, and other setup and maintenance tasks. These are easier and more pleasant to use than devices that require setup and control from a computer.

Finding Supported Printers and Scanners

Hewlett-Packard (HP) printers and multifunction devices have excellent Linux support via the *hplip*, *hplip-hpijs*, *hplip-sane*, and *hplip-scan-utils* packages. Of course every Linux is special and has different package names, such as *hpijs-ppds*, *hplip-data*, *printer-driver-hpcups*, *hplip-common*, and *libsane-hpaio*. Searching for *hplip* should find them.

Not all HP printers and MFDs are supported in Linux; see the link below for HP's Linux support database.

Brother has good machines, decent customer support, and good prices on inks. They have both machines with native Linux support and some that require the Brother drivers.

Canon, Epson, Honeywell, Fujitsu, IBM, Lexmark, Kodak, Tektronix, Samsung, Sharp, Xerox, Toshiba, and many other brands have some level of Linux support. It can be difficult to learn which models are supported. Some vendors tell you in their product specs. There are several websites to check, though they are usually not complete nor up-to-date, but they're good places to start:

- Supported HP printers
- OpenPrinting.org printer listings
- H-node printers and MFDs
- ThinkPenguin store
- Ubuntu page for supported printers
- IPP Everywhere Printers

CUPS Printer Drivers

Linux printer drivers are provided by CUPS, the Common Unix Printing System. CUPS has been the standard printing subsystem for Linux since around 2000. Apple started using CUPS around 2002, then hired CUPS creator Michael Sweet and bought the source code in 2007. Sweet left Apple in 2019, and Apple's involvement stalled—

see apple/cups on GitHub. Mr. Sweet has not been idle; rather, he has been hard at work on the OpenPrinting.org's CUPS fork at OpenPrinting/cups on GitHub.

There is a lot more to CUPS than writing code. Michael Sweet, Till Kamppeter, and others invested a lot of work into getting manufacturers on board and developing common printing standards and APIs. The centers of CUPS and printing standards development are The Printer Working Group and OpenPrinting.

Printer drivers in CUPS consist of one or more filters specific to a printer, which are packaged in PPD (PostScript Printer Description) files. All printers in CUPS, even non-PostScript printers, need a PPD. The PPDs contain descriptions about the printers, printer commands, and filters.

Filters translate print jobs to formats the printer can understand, such as PDF, HP-PCL, raster, and image files, and they pass in commands for operations such as page selection, paper size, colors, contrast, and media type. PPDs are plain-text files, and the PPDs for all supported printers are in */usr/share/cups/model/*. Installed printers have PPDs in */etc/cups/ppd/*.

PPDs Are Doomed

CUPS has relied on PPDs from its inception, and they have worked well. However, there is new approach in development called *driverless* printing. Instead of using static PPD files, the printer advertises its capabilities and does not require driver installation on client machines. The idea is to make connecting to a new printer as easy as connecting to a new network with NetworkManager, which automatically finds available networks and does not require you to install drivers or to manually configure every new network. This is especially advantageous for mobile devices like phones and tablets, which have limited storage space and limited screen size for hassling with driver installation.

Driverless was introduced in CUPS 2.2.0. You should have at least CUPS 2.2.4 (released in June 2017) for reliable performance. You can learn more at the following resources:

- Printer Applications: A New Way to Print in Linux
- CUPS Driverless Printing

15.1 Using the CUPS Web Interface

Problem

You need to find the CUPS administration tool.

Solution

Open the CUPS web interface in your web browser at *http://localhost:631/* (Figure 15-1).

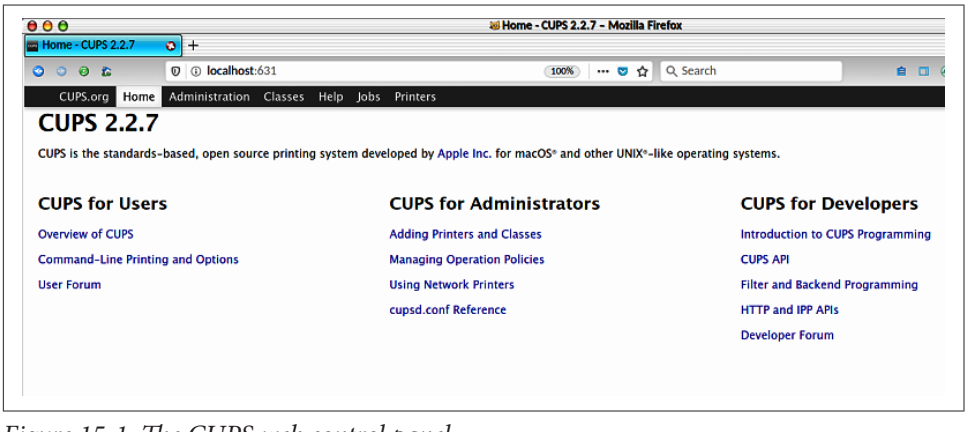


Figure 15-1. The CUPS web control panel

Discussion

There are numerous graphical tools for managing printers, such as *system-config-printer* and the YaST printer module in openSUSE. The CUPS web administration page provides the most complete management options and is the same on all Linux distributions.

See Also

- CUPS documentation

15.2 Installing a Locally Attached Printer

Problem

You need to install a new printer which is connected to your PC. You have wisely chosen a printer with native CUPS support.

Solution

Use the CUPS web control panel. Your printer should be connected and powered on. The following examples are on a Linux Mint system.

Go to the Administration tab and click Add Printer. It will ask you for your login (Figure 15-2). (If your login does not work, and only the root login succeeds, see Recipe 15.7 to learn how to configure CUPS to accept nonroot logins.) Check "Save debugging information for troubleshooting," and check "Share printers connected to this system" to share printers directly attached to your computer. This only enables sharing, and then you must enable sharing on each printer you want to share.

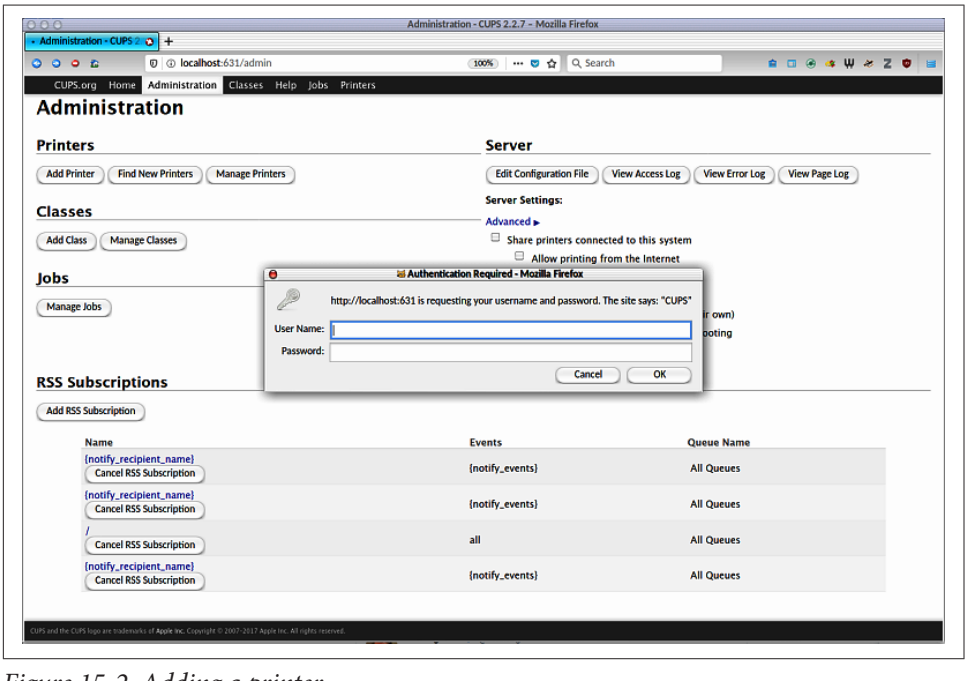


Figure 15-2. Adding a printer

On the next screen, CUPS discovers and lists your printer in the Local Printers section (Figure 15-3). Select your printer and click Continue.

Now you should see a screen like Figure 15-4, with Name, Description, and Location fields. The Name and Description fields are automatically filled, and you may change these to whatever you want. The Name field appears in the printer dialogs when you print a document.

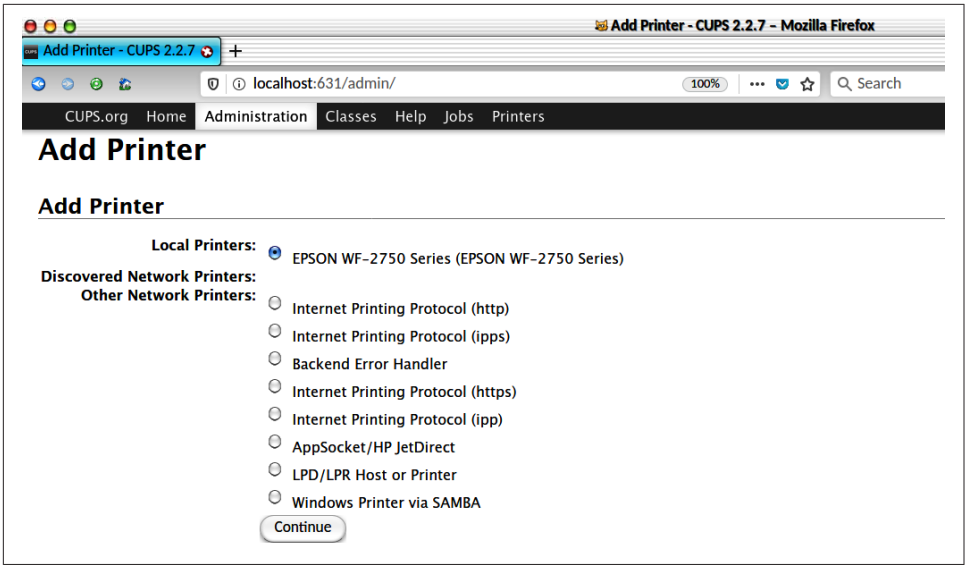


Figure 15-3. CUPS finds your local printer

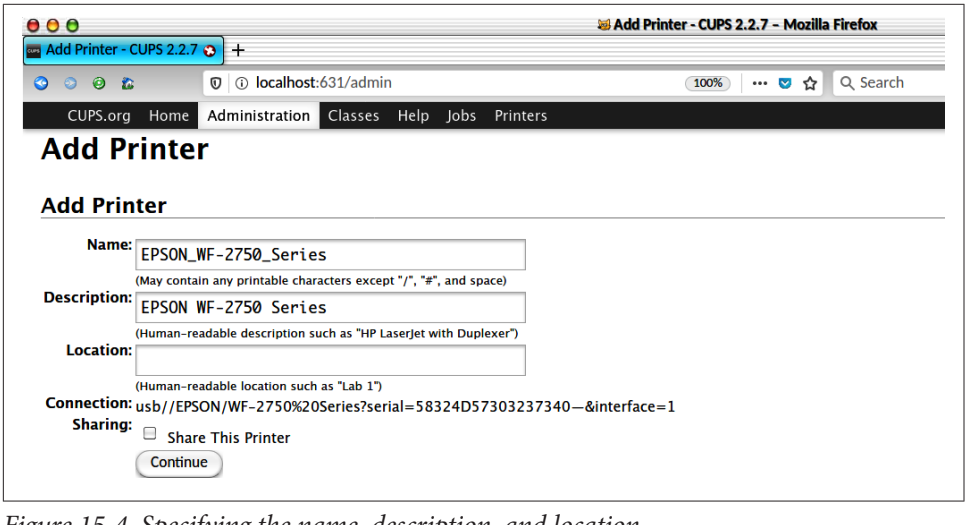


Figure 15-4. Specifying the name, description, and location

Select your printer driver. CUPS displays a giant list for you to choose from. Find the driver for your printer model. In Figure 15-5 the drivers come from the *epson-inkjet-printer-escpr* package (*printer-driver-escpr* on Ubuntu), for Seiko Epson color ink jet printers.

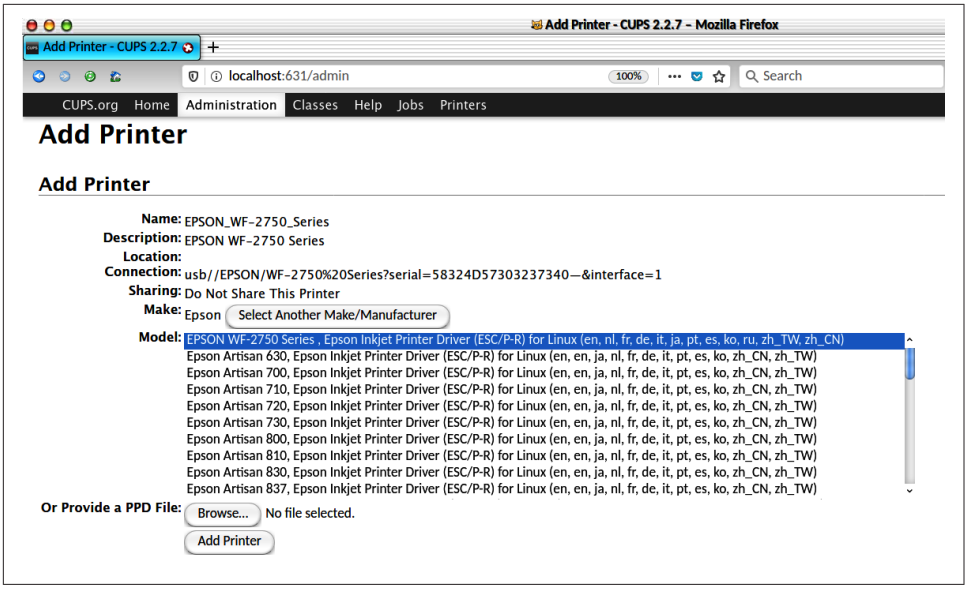


Figure 15-5. Select the printer driver

The final configuration screen is for setting default options, such as paper type, paper size, color or black and white, print quality, and other options according to what your printer and printer driver support. When you are finished, click Set Default Options (Figure 15-6).

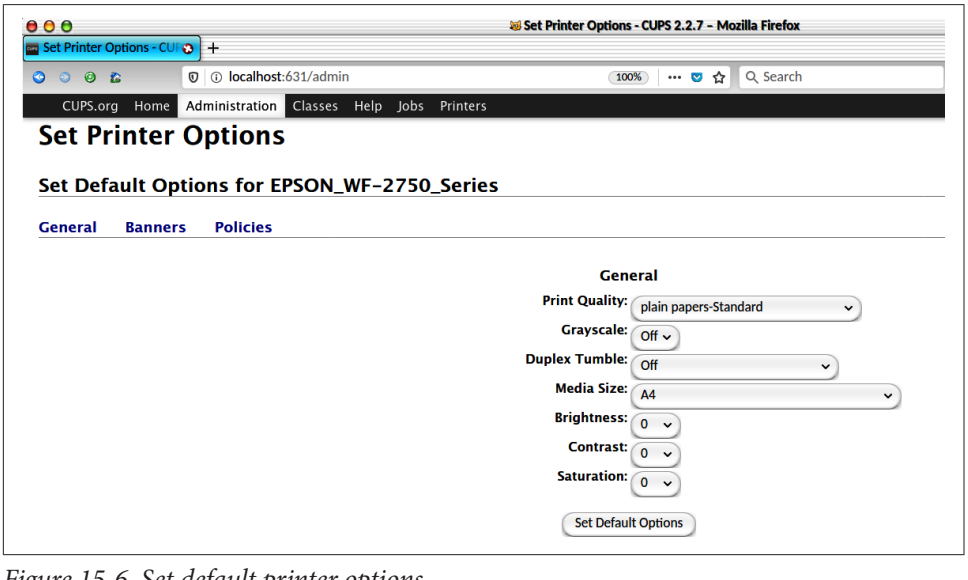


Figure 15-6. Set default printer options

When you are finished, you see the Printers page, listing all of your installed printers (Figure 15-7).

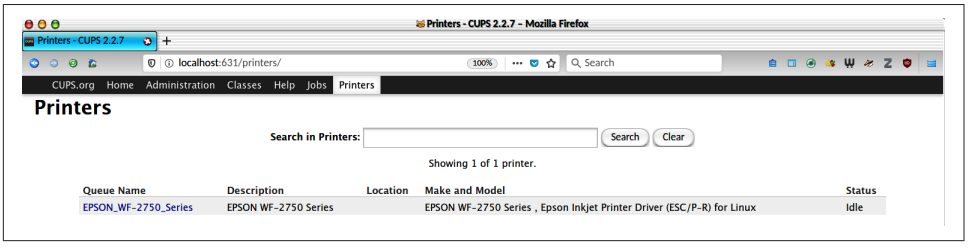


Figure 15-7. All installed printers

Click on your new printer, and print a test page from the Maintenance drop-down menu. When it prints correctly, you are done.

Discussion

In Figure 15-2 you see two buttons, Add Printer and Find New Printers. These are pretty much the same thing, with the discovered printer listings organized differently.

You will probably have more than one printer driver to choose from; for example, it is common to see both CUPS+Gutenprint and Foomatic drivers for the same printer. Gutenprint used to be the better choice for color printers; try both to see which you prefer. CUPS+Gutenprint Simplified drivers contain fewer features and options than the full versions.

See Also

- CUPS documentation

15.3 Giving Printers Useful Names

Problem

When you open the printer dialog in a document, you have several printers to choose from, including some that look similar, and you aren't sure which one to use.

Solution

When you install a printer, enter a descriptive name in the Name field (Figure 15-8). You have to do this at installation because you cannot change the name after installation.

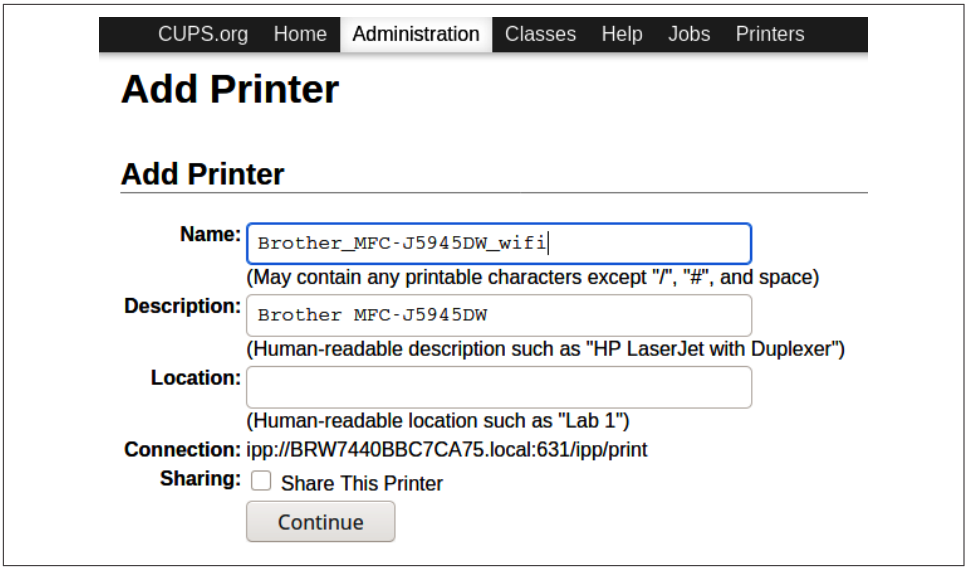


Figure 15-8. Use the printer name to identify your printers

Discussion

When you first install a printer using the CUPS web control panel, there are Description and Location fields that you can use, and these show up in the CUPS web control panel. But many apps that you print from do not read these fields and only read the Name field. Some exceptions are the Evolution mail client and the Firefox and Chromium web browsers, which display the name, location, and status.

See Also

- CUPS documentation

15.4 Installing a Network Printer

Problem

There is a shared network printer on your network, and you want to install it on your computer.

Solution

The procedure is the same as installing a locally attached USB printer (Recipe 15.2), except you select a discovered network printer. The printer must be powered on and

on the same network segment as your computer. You will see it listed under Discovered Network Printers (Figure 15-9).

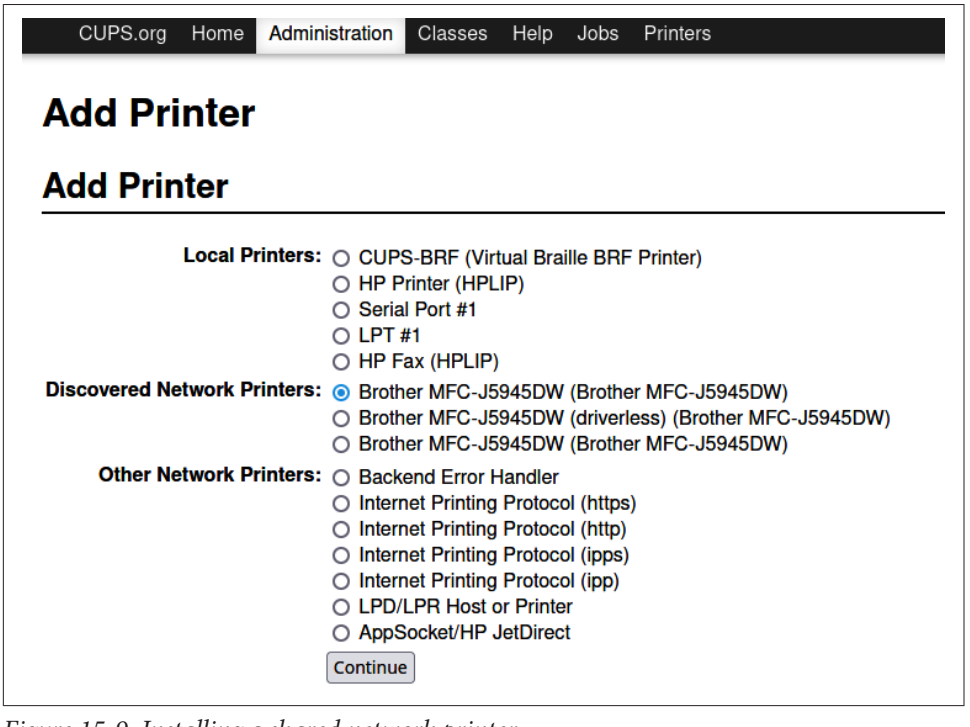


Figure 15-9. Installing a shared network printer

You need TCP port 631 open in the firewalls on all clients.

Discussion

What if CUPS does not discover your printer? See Recipe 15.11 for some troubleshooting help. If CUPS does not see your printer, you cannot install it.

See Also

- CUPS documentation

15.5 Using Driverless Printing

Problem

Your printer is not supported in CUPS, and you want to try the driverless printing option. Or, you want to connect your Android or iOS devices to your printer.

Solution

You may have already seen driverless options in the CUPS printer driver selector. The following example is for my Brother MFC-J5945DW, which does not have native CUPS support.

Go to Administration → Add Printer in the CUPS web control panel. CUPS sees my Brother machine in Discovered Network Printers (Figure 15-10). There is a *driverless* option, and that is the correct one to choose.

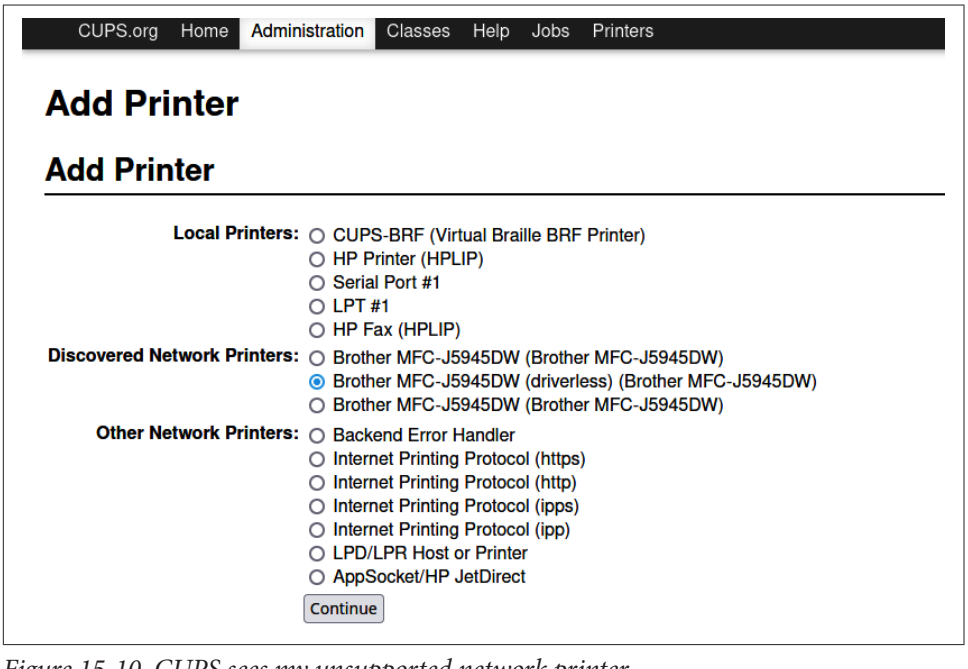


Figure 15-10. CUPS sees my unsupported network printer

Continue the installation, and select the correct driver, which in Figure 15-11 is the "Brother MFC-J5945DW, driverless, cups-filters 1.25.0 (en)."

Print a test page, and if it looks right, the installation is completed.

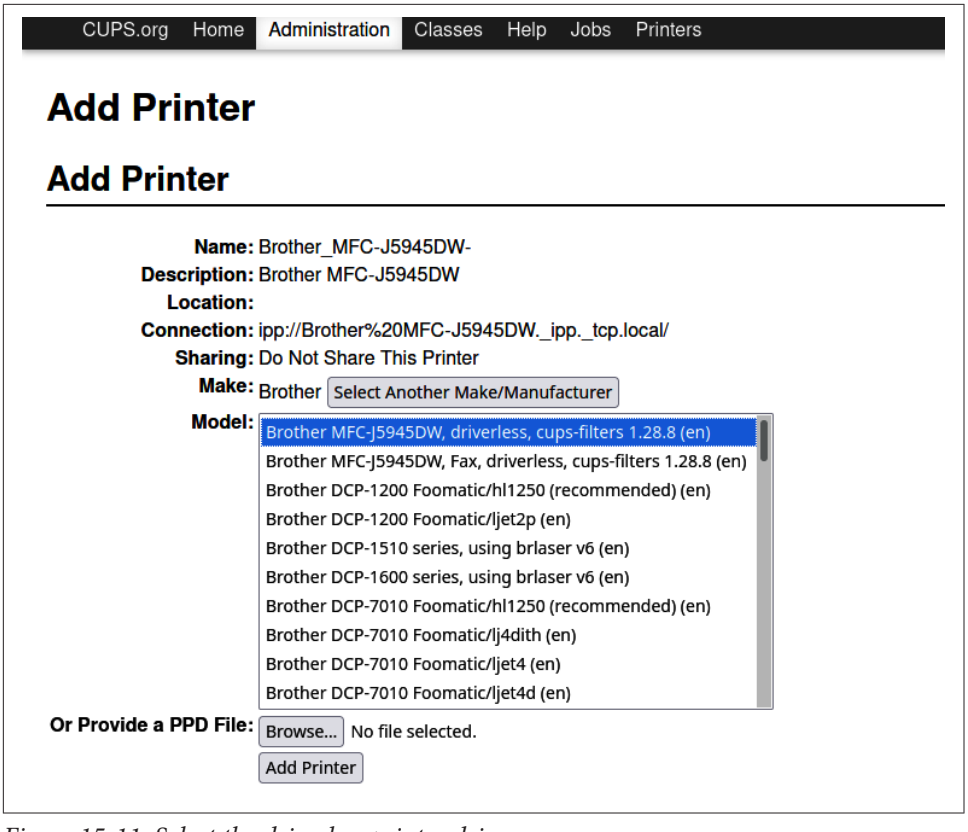


Figure 15-11. Select the driverless printer driver

Discussion

Strictly speaking, this isn't driverless, because CUPS creates PPD files in */etc/cups/ppd* for your "driverless" printer. However, you don't have to maintain directories full of OpenPrinting.org and Gutenprint PPDs.

Your printer must support driverless printing, which means it must support the Mopria, AirPrint, IPP Everywhere, or WiFi Direct Print standard. These are all similar: the printer advertises itself, its network address, and basic functionality via the Avahi daemon. Avahi provides service discovery on your local network, using the mDNS/DNS-SD protocol suite. (Apple calls this service Bonjour and Zeroconf.)

Driverless printing in CUPS works great for Android and iOS devices. You only need to install a printer app. If your printer is driverless enabled, and especially if it is Mopria certified, then your mobile devices will have no trouble finding it. Mopria certification means your printer supports wireless printing sent from mobile devices. If

your printer documentation does not tell you if it is Mopria certified, run the following command, which shows that the printer is Mopria certified:

```
$ avahi-browse -rt _ipp._tcp
[...]
txt = ["mopria-certified=1.3"
[...]
```

See Also

- Debian Wiki, Driverless Printing
- CUPS documentation

15.6 Sharing Nonnetworked Printers

Problem

You want to share a printer that does not have built-in networking.

Solution

CUPS shares printers that do not have networking, but are connected to a PC on your network. First, you must have your name services working so that all your LAN hosts can ping each other.

Make sure that printer sharing is enabled on the Administration screen by checking "Share printers connected to this system." Then enable sharing on the printer you want to share (Figure 15-12).

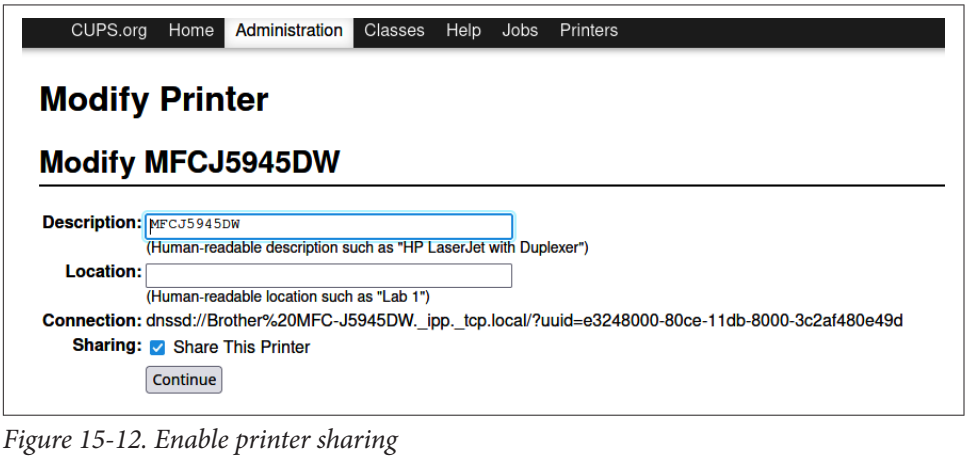


Figure 15-12. Enable printer sharing

CUPS will advertise the printer to your network. Any Linux client on your network who wants to use this printer must install it the same way as installing a network or locally attached printer; start at Administration → Add Printer, then go through the normal installation process.

You may also share with Windows and macOS clients. macOS has native support for discovery via DNS-SD/mDNS and IPP. DNS-SD/mDNS is provided by Avahi on Linux and is called Bonjour on the Macintosh. Use the Macintosh control panel to find and install shared CUPS printers.

Windows 10 natively supports DNS-SD/mDNS. Older Windows releases support sharing via the Internet Printing Protocol (IPP). Use the Windows printer control panel to find and install shared CUPS printers.

Discussion

In olden times, before network-enabled printers were common and inexpensive, admins used dedicated printer servers. These were old PCs, old laptops, small single-board computers, or commercial printer server devices. You can still buy small printer server gadgets, and they cost a lot less than in the old days.

Nowadays most printers have networking built in and are simpler to manage.

See Also

- CUPS documentation

15.7 Correcting the "Forbidden" Error Message

Problem

When you try to perform any administrative task in the CUPS web control panel, for example, adding a new printer, your login fails and there is a "Add Printer Error Unable to add printer: Forbidden" message.

Solution

On some Linux distributions, such as openSUSE, the default configuration allows only the root user to do CUPS administration tasks. Edit */etc/cups/cups-files.conf* to allow nonroot users to do CUPS administration tasks. Look for these lines:

```
# Administrator user group, used to match @SYSTEM in cupsd.conf policy rules...
# This cannot contain the Group value for security reasons...
SystemGroup root
```

This is why only the root login works. You can add your own private user group, like our example user Duchess, whose private group is *duchess*:

```
SystemGroup root duchess
```

After saving changes to */etc/cups/cups-files.conf*, restart the CUPS service:

```
$ sudo systemctl restart cups.service
```

Now Duchess can do CUPS administration tasks.

Another approach is to use a system group created for this purpose. On Ubuntu Linux distributions, this is the *lpadmin* group, and Fedora uses the *sys* and *wheel* groups. You may create your own group for CUPS administration, like the following example that creates a *cupsadmin* group, and adds the user Mad Max to this group:

```
$ sudo groupadd -r cupsadmin
$ sudo usermod -aG cupsadmin madmax
```

Mad Max must log out, then log back in to activate the new group membership. Add the *cupsadmin* group to *SystemGroup* in */etc/cups/cups-files.conf*:

```
SystemGroup root duchess cupsadmin
```

Restart CUPS, and Mad Max can go to work.

Discussion

There should also be a section like this in */etc/cups/cups-files.conf*:

```
# Default user and group for filters/backends/helper programs; this cannot be
# any user or group that resolves to ID 0 for security reasons...
#User lp
#Group lp
```

None of the groups listed for *SystemGroup* can match *Group*. If you try to use *lp*, as in the preceding example, CUPS will not start, and you will see error messages in */var/log/cups/error_log* or the syslog, depending how yours is configured in */etc/cups/cups-files.conf*.

If your Linux uses SysV init instead of systemctl, restart it with this command:

```
$ sudo /etc/init.d/cups restart
```

See Also

- CUPS documentation
- Chapter 4

15.8 Installing Printer Drivers

Problem

You need to know if installing CUPS also installs a complete set of printer drivers, or if there are more that you might want that are not included with CUPS.

Solution

Most Linuxes install a subset of all available printing options. Every Linux distribution varies in which printer drivers are available, which ones are included in a default installation, and which specific names are used for packages, especially between Ubuntu and everyone else.

The following list comprises a basic set of CUPS packages and printer drivers:

- *cups* (server and client)
- *cups-filters* (OpenPrinting CUPS filters and backends)
- *gutenprint* (Gutenprint printer drivers)
- *foomatic* (Foomatic printer drivers)
- OpenPrinting.org PPDs; for example, OpenSUSE provides:
  - *OpenPrintingPPDs*
  - *OpenPrintingPPDs-ghostscript* (interpreter of printer drivers written in the PostScript language)
  - *OpenPrintingPPDs-hpijs* (HP printers)
  - *OpenPrintingPPDs-postscript*
- *cups-client* (command-line utilities for setting up and managing printers)

OpenPrinting.org includes Foomatic. Fedora and Ubuntu ship *foomatic* packages, while OpenSUSE includes *OpenPrinting* packages. The names change, but the functionality is the same.

These packages may provide everything you need. The following are additional printer packages you might find useful:

- *gimp-gutenprint* (provides a more featureful printer dialog for GIMP, the GNU Image Manipulation Program)
- *bluez-cups* (connect Bluetooth printers)
- *cups-airprint* (share printers with iOS devices)
- *ptouch-driver* (Brother P-touch label printers)

- *rasterview* (view Apple raster images such as GIF, JPEG, and PNG, see MSweet.org/rasterview
- *c2esp* (some Kodak all-in-one printers)

Ubuntu packages the largest assortment of printer drivers. Many, but not all, of the package names start with *printer-driver*:

- *openprinting-ppds* (OpenPrinting printer support, PostScript PPD files)
- *printer-driver-all* (printer drivers metapackage)
- *printer-driver-brlaser* (some Brother laser printers)
- *printer-driver-c2050* (Lexmark 2050 Color Jetprinter)
- *printer-driver-foo2zjs* (ZjStream-based printers)
- *printer-driver-c2esp* (Kodak ESP AiO color inkjet series)
- *printer-driver-cjet* (Canon LBP laser printers)
- *printer-driver-cups-pdf* (PDF writing via CUPS)
- *printer-driver-dymo* (DYMO label printers)
- *printer-driver-escpr* (Epson Inkjets that use ESC/P-R)
- *printer-driver-foo2zjs* (ZjStream-based printers)
- *printer-driver-fujixerox* (Fuji Xerox printers)
- *printer-driver-gutenprint* (printer drivers for CUPS)
- *printer-driver-hpcups* (HP Linux Printing and Imaging, CUPS Raster driver (hpcups))
- *printer-driver-hpijs* (HP Linux Printing and Imaging, printer driver (hpijs))
- *printer-driver-indexbraille* (CUPS printing to Index Braille printers)
- *printer-driver-m2300w* (Minolta magicolor 2300W/2400W color laser printers)
- *printer-driver-min12xxw* (KonicaMinolta PagePro 1[234]xxW)
- *printer-driver-oki* (OKI Data printers)
- *printer-driver-pnm2ppa* (HP-GDI printers)
- *printer-driver-postscript-hp* (HP Printers PostScript Descriptions)
- *printer-driver-ptouch* (printer driver for Brother P-touch label printers)
- *printer-driver-pxljr* (HP Color LaserJet 35xx/36xx)
- *printer-driver-sag-gdi* (Ricoh Aficio SP 1000s/SP 1100s)
- *printer-driver-splix* (Samsung and Xerox SPL2 and SPLc la)

Discussion

If you don't see your printer listed in the driver selector in the CUPS web interface, try searching for the brand name of your printer in your package manager. This may all seem rather messy, but setting up printers is untidy on all computing platforms (not just on Linux).

See Also

- CUPS documentation
- Ghostscript documentation
- OpenPrinting
- The Printer Working Group

15.9 Modifying an Installed Printer

Problem

You want to change the configuration of a printer that is already installed. For example, you want to share it.

Solution

Open the printer in the CUPS web control panel, then click Administration → Modify Printer. This is similar to installing a new printer, except it displays the printer's current settings. Figure 15-13 enables sharing the printer. (Note that sharing must first be enabled on the Administration → Advanced page.)

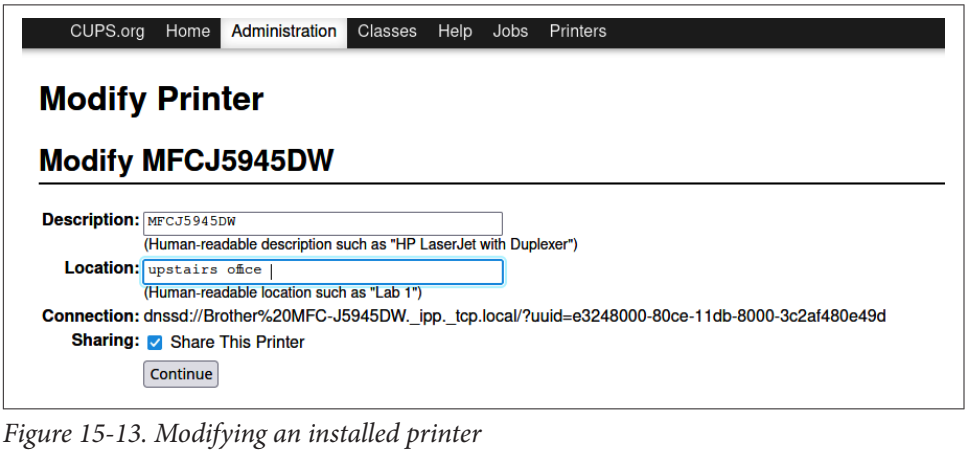


Figure 15-13. Modifying an installed printer

Discussion

You can change everything except the printer name.

See Also

- CUPS documentation

15.10 Saving Documents by Printing to a PDF File

Problem

You want to save a web page, or any document, to a PDF file instead of sending it to a printer.

Solution

Look at the File → Print dialog in any application, and you will see an option to print to a PDF file (Figure 15-14).

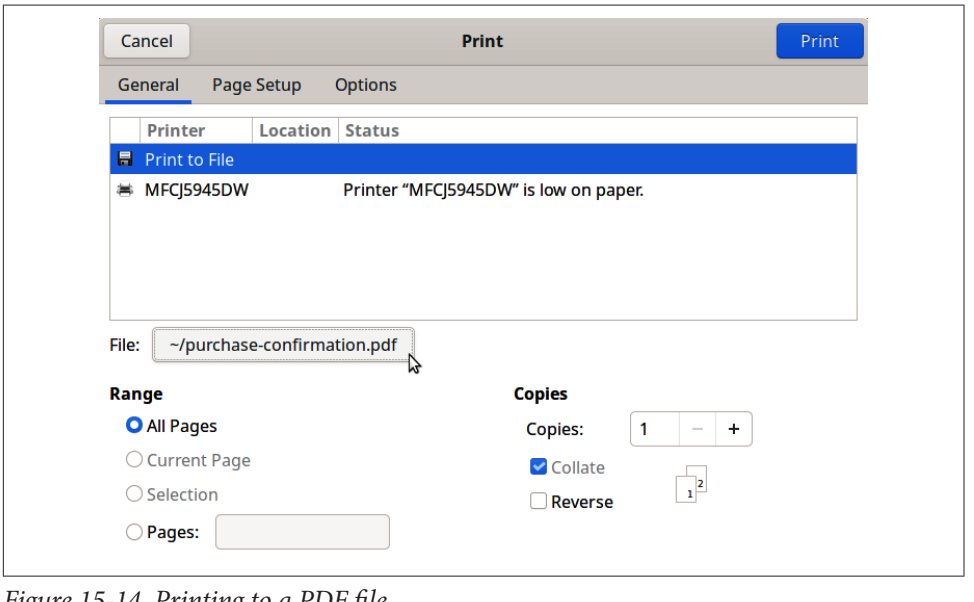


Figure 15-14. Printing to a PDF file

You have all the usual options, such as filename and location, margins, print quality, color or monochrome, and page orientation. The printer dialogs look different in different apps; for example, the Firefox web browser printer dialog includes a document preview. In other apps the preview is usually a separate button.

Discussion

Print to file is great for saving web confirmation forms and receipts, and creating PDFs from any type of documents.

See Also

- CUPS documentation

15.11 Troubleshooting

Problem

Printing doesn't work! How do you fix it?

Solution

These are the most common issues with printers on Linux:

- For shared printers, make sure that your networking is set up correctly and your firewall allows TCP port 631. If you have more than one network, verify that the printer is on the same network as your computer.
- For USB-connected printers, try a different USB port or a different cable.
- Verify you are using the correct printer drivers, or try driverless.
- The CUPS daemon is managed by systemd. Try restarting the daemon:

  ```
  $ sudo systemctl restart cups.service
  ```

  Or reboot, and also power-cycle your printer.
- Check your log files on the CUPS web administration page; you can view both the error log and the access log. Turn the logging level up to Debug to get the most information. (Click Edit Configuration File, then set *LogLevel debug*.)

Discussion

The most important factor is using printers with good Linux support. This prevents most problems.

See Also

- CUPS documentation

CHAPTER 16

Managing Local Name Services with Dnsmasq and the hosts File

Dnsmasq is an excellent server for LAN name services, both Domain Name System (DNS) and Dynamic Host Discovery Protocol (DHCP). Dnsmasq also provides BOOTP, PXE, and TFTP, which are for network booting and installing operating systems from a network server. Dnsmasq supports IPv4 and IPv6, provides local DNS caching, and acts as a stub resolver.

This chapter covers setting up local DNS and DHCP using Dnsmasq and the */etc/hosts* file together. */etc/hosts* is the very old way of setting up DNS, mapping hostnames to IP addresses in a static file. */etc/hosts* by itself is sufficient for very small networks.

Dnsmasq is designed for LAN name services. It is lightweight and simple to configure, especially in comparison to BIND, the dominant DNS server, which is heavyweight and has a rather steep learning curve.

Dnsmasq and */etc/hosts* work great together. Dnsmasq reads the entries in */etc/hosts* into DNS.

The DHCP server in Dnsmasq automatically integrates with DNS. All you need for Dnsmasq to create DNS entries for your DHCP clients is to configure your DHCP clients to send their hostnames to the DHCP server, which is the default in most Linux distributions.

There are four types of DNS servers: recursive resolvers, root name servers, top-level domain (TLD) name servers, and authoritative name servers.

Recursive resolvers answer DNS requests. A stub resolver, like Dnsmasq and systemd-resolved, forwards any requests it cannot answer from its cache to an upstream resolver. When you visit a website a recursive resolver finds the site's DNS

information by querying the other three types of DNS servers. Recursive resolvers cache this information to make it available more quickly. Your ISP's name servers, and services like OpenDNS, Cloudflare, and Google Public DNS are all recursive resolvers.

There are 13 types of root name servers, spread all over the planet, and there are currently several hundred root name servers. A root server accepts a query from a recursive resolver, then the root server directs the request to the appropriate TLD server according to the top-level domain: .com, .net, .org, .me, .biz, .int, .biz, .gov, .edu, and so on. The Internet Corporation for Assigned Names and Numbers—ICANN—oversees all of these servers and domains.

Authoritative name servers are the source records for a domain and are controlled by the owner of the domain. Dnsmasq can serve as your authoritative name server, though I recommend using BIND. See the Authoritative Configuration section of *man 8 dnsmasq* to learn more.



Too Many Name Service Utilities

Linux distributions are still transitioning to NetworkManager and *systemd-resolved* from the legacy *resolvconf*, which has long been the default DNS resolver on Linux systems. This presents a bit of a tangle for Linux users, with continual changes and the various distros making the transition at different rates. Pay close attention to the documentation, forums, and release notes for your particular Linux.

It should be possible to use Dnsmasq as the DNS backend for NetworkManager, because NetworkManager has a plug-in for this. But on some Linux distributions, this does not yet work correctly (Recipe 16.5).

You don't need systemd-resolved running on your Dnsmasq server because it will contend with Dnsmasq for control of the system's stub DNS resolver.

By the time you read this, all of this may be different, but for now the recipes aim to be reliable rather than cutting edge.

16.1 Simple Name Resolution with /etc/hosts

Problem

You want a simple, fast way to set up name resolution without having to run a DNS server.

Solution

This is what the */etc/hosts* file is made for. Your LAN computers must have static IP addresses. The following is an example for three computers:

```
127.0.0.1 localhost
::1 localhost ip6-localhost ip6-loopback
192.168.43.81 host1
192.168.43.82 host2
192.168.43.83 host3
```

Copy these entries to all three hosts, then try pinging each other by hostnames, like this example for pinging *host2* from *host3*:

```
host3:~$ ping -c2 host2
PING host2 (192.168.43.82) 56(84) bytes of data.
64 bytes from host2 (192.168.43.82): icmp_seq=1 ttl=64 time=3.00 ms
64 bytes from host2 (192.168.43.82): icmp_seq=2 ttl=64 time=3.81 ms

--- host2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 3.001/3.403/3.806/0.402 ms
```

/etc/hosts also manages domain names, so you can give your LAN a cool domain. In the following examples, that is *sqr3l.nut*. First enter the IP address, then the fully qualified domain name (FQDN), then the hostname:

```
127.0.0.1 localhost
::1 localhost ip6-localhost ip6-loopback
192.168.43.81 host1.sqr3l.nut host1
192.168.43.82 host2.sqr3l.nut host2
192.168.43.83 host3.sqr3l.nut host3
```

Now your hosts can connect to each other with their hostnames, like *host1*, or their FQDNs, like *host1.sqr3l.nut*.



Shared and Individual Hosts Entries

You can have both shared and private entries in */etc/hosts*. Anything you want shared must be copied to all the relevant hosts. Anything else in your hosts file that is not copied to other hosts will work only for you. See Recipe 16.2 to learn more.

Discussion

127.0.0.1 localhost and *::1 localhost ip6-localhost ip6-loopback* are required. Yours might look a little different, but however they are written, do not delete them. They are assigned to the loopback device, a special virtual network interface that your Linux system uses to communicate with itself.

You can ping them and use them to connect to local servers. For example, when you use the CUPS web administration page, you are using the loopback device. Enter *127.0.0.1:631* or *localhost:631* to open it (Figure 16-1).

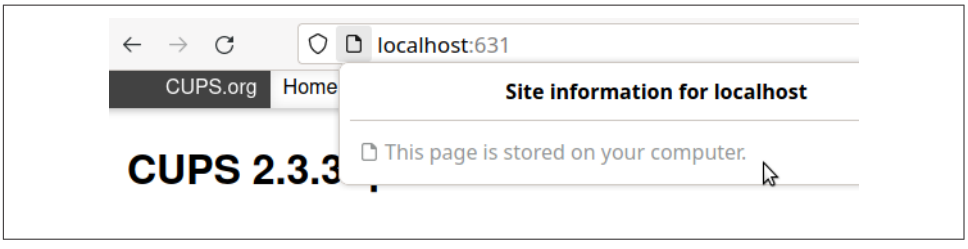


Figure 16-1. Opening a local web page with the loopback device

The virtual network interface for the loopback device is *lo*. Use the *ip* command to see it:

```
$ ip addr show dev lo
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
  default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
```

Your system does not need a physical network interface for the loopback device to work.

Use the *hostname* command to confirm that your configuration is correct. Check the hostname of your computer:

```
$ hostname
host1
```

Check the FQDN:

```
$ hostname -f
host1.sqr3l.nut
```

Check the domain name:

```
$ hostname -d
sqr3l.nut
```

/etc/hosts does not scale well, but for small networks it may be all you ever need for your local DNS.

See Also

- *man 5 hosts*
- *man 8 ping*
- Recipe 16.2

16.2 Using /etc/hosts for Testing and Blocking Annoyances

Problem

You are working on development servers, and you want to manage their DNS without hassles. Or, you want a simple way to block annoying sites.

Solution

Suppose the name of a dev server you are working on is *dev.stashcat.com*. Make an entry for it your */etc/hosts* file:

```
192.168.10.15 dev.stashcat.com
```

You don't have to bother your network admin or mess with your DNS server, but can create and remove entries in */etc/hosts* as you need.

Another fun trick is to map annoying websites to bogus IP addresses:

```
12.34.56.78  badsite.com
12.34.56.78  www.badsite.com
```

This makes the site unreachable from your computer. Most how-tos use the loopback address, 127.0.0.1, and it works, but I prefer to keep the annoying sites separate. You can use the same fake IP address for multiple annoying sites.

If your web browser still reaches the site after adding it to */etc/hosts*, clear your browser cache and try again.

Discussion

When you run a LAN Dnsmasq server, keep in mind that all entries in */etc/hosts* on the Dnsmasq server will be applied to all Dnsmasq clients, so don't put your name server on your development computer.

Linux has a number of DNS managers, and */etc/hosts* is read first. The order is set in the */etc/nsswitch.conf* file on the *hosts* line. The following example is from Ubuntu 20.04:

```
hosts: files mdns4_minimal [NOTFOUND=return] dns mymachines
```

files is */etc/hosts*.

mdns4_minimal uses the Avahi autodiscovery service to locate network services.

[NOTFOUND=return] means that if *mdns4_minimal* is working but the requested host is not found, the DNS lookup should stop and return an error. If the *mdns4_minimal* service is not found, continue the lookup.

dns uses any available DNS server.

mymachines refers to the *systemd-machined* service, which tracks local virtual machines and containers.

You should put *files dns* first on your Dnsmasq server.

See Also

- *man 5 hosts*
- *man 5 nsswitch.conf*
- *man 8 systemd-machined.service*

16.3 Finding All DNS and DHCP Servers on Your Network

Problem

You want to know if there are any DNS and DHCP servers on your LAN, other than your Dnsmasq server.

Solution

Probe your LAN with *nmap*. The following example searches the local network for all open TCP ports and finds an open TCP port 53, which is used by DNS. This is indicated by "53/tcp open domain":

```
$ sudo nmap --open 192.168.1.0/24
Starting Nmap 7.70 ( https://nmap.org ) at 2021-05-23 13:25 PDT
[...]
Nmap scan report for dns-server.sqr3l.nut (192.168.1.10)
Host is up (0.12s latency).
Not shown: 998 filtered ports
Some closed ports may be reported as filtered due to --defeat-rst-ratelimit
```

```
PORT   STATE SERVICE
22/tcp open  ssh
53/tcp open  domain
[...]

Nmap done: 256 IP addresses (3 hosts up) scanned in 81.38 seconds
```

By default, *nmap* only looks for TCP ports. DNS servers listen to TCP and UDP 53, and DHCP listens on UDP 67. The following example looks only for ports UDP 53 and 67:

```
$ sudo nmap -sU -p 53,67 192.168.1.0/24
Starting Nmap 7.80 ( https://nmap.org ) at 2021-05-27 18:05 PDT

Nmap scan report for dns-server.sqr3l.nut (192.168.1.10)
Host is up (0.085s latency).

PORT   STATE         SERVICE
53/udp open          domain
67/udp open|filtered dhcps

Nmap done: 256 IP addresses (3 hosts up) scanned in 13.85 seconds
```

nmap found one DNS/DHCP server, on dns-server.sqr3l.nut.

The following command searches for all open TCP and UDP ports on the network:

```
$ sudo nmap -sU -sT 192.168.1.0/24
```

This takes several minutes to complete, and then you have a list of the active services on all up hosts in your network, including any services running on nonstandard ports.

Discussion

Be very careful with port scanning, and use it only on networks that you have permission for. Port scanning other networks is often treated as a hostile act, like you are probing for vulnerabilities to exploit.

Multiple name servers can cause conflicts, and in any case it is good to know if your users are running any servers.

On most Linux systems, the package to install is *nmap*.

See also

- *man 1 nmap*

16.4 Installing Dnsmasq

Problem

You want to install Dnsmasq and take care of any prerequisites.

Solution

Install the *dnsmasq* package. In this recipe the Dnsmasq server is named *dns-server*. You will use both Dnsmasq and the */etc/hosts* file to configure your DNS server.

After installation, stop Dnsmasq if it is running:

```
$ systemctl status dnsmasq.service
● dnsmasq.service - dnsmasq - A lightweight DHCP and caching DNS server
     Loaded: loaded (/lib/systemd/system/dnsmasq.service; enabled; vendor
     preset: enabled)
     Active: active (running) since Mon 2021-05-24 05:49:36 PDT; 6h ago
[...]
$ sudo systemctl stop dnsmasq.service
```

Give your Dnsmasq server a static IP address, if it does not already have one. Do this with NetworkManager's graphical control panel (*nm-connection-editor*), or with the *nmcli* command.

The following example uses *nmcli* to find your active connections:

```
$ nmcli connection show --active
NAME       UUID                   TYPE      DEVICE
1local     3e348c97-4c5f-4bbf-967e  wifi      wlan1
1wired     0460d735-e14d-3c3f-92c0  ethernet  eth1
```

Then assign the static IP address you want your DNS server to use, using the NAME to identify the correct connection:

```
$ nmcli con mod "1wired" \
  ipv4.addresses "192.168.1.30/24" \
  ipv4.gateway "192.168.1.1" \
  ipv4.method "manual"
```

Then restart NetworkManager:

```
$ sudo systemctl restart NetworkManager.service
```

Next, check if your Linux is running *systemd-resolved.service*:

```
$ systemctl status systemd-resolved.service
```

If it is, see Recipe 16.5 before configuring Dnsmasq, and also to learn how to configure NetworkManager on your Dnsmasq server.

Discussion

systemd is implemented differently amongst the various Linuxes. For example, openSUSE Leap 15.2 does not use the *systemd-resolved.service*, so you should not have to make any systemd changes to enable Dnsmasq to control your LAN DNS. Fedora 33 and up, and Ubuntu 17.04 and up, run *systemd-resolved.service*, and it should be disabled on your Dnsmasq server.

See Also

- Recipe 16.5
- Dnsmasq

16.5 Making systemd-resolved and NetworkManager Play Nice with Dnsmasq

Problem

systemd-resolved and NetworkManager are conflicting with Dnsmasq, and you want them out of the way.

Solution

Check if the *systemd-resolved.service* is running:

```
$ systemctl status systemd-resolved.service

● systemd-resolved.service - Network Name Resolution
     Loaded: loaded (/usr/lib/systemd/system/systemd-resolved.service; enabled;
     vendor preset: enabled)
     Active: active (running) since Sat 2021-05-22 12:57:34 PDT; 1min 21s ago
[...]
```

That shows that it is running. *systemd-resolved.service* is fine for providing a stub DNS resolver for client machines, but not DNS servers. Disable it:

```
$ sudo systemctl stop systemd-resolved.service
$ sudo systemctl disable systemd-resolved.service
```

Then look at */etc/resolv.conf*, which should be a symlink:

```
$ ls -l /etc/resolv.conf
lrwxrwxrwx 1 root root 39 May 21 20:38 /etc/resolv.conf ->
   ../run/systemd/resolve/stub-resolv.conf
```

When it is a symlink, it is managed by *systemd-resolved.service*. To remove control from *systemd-resolved.service*, delete the symlink and create a plain-text file with the same name:

```
$ sudo rm /etc/resolv.conf
$ sudo touch /etc/resolv.conf
```

Now that */etc/resolv.conf* is a file and not a symlink, it is managed by NetworkManager. Open your NetworkManager configuration file and look for the *[main]* section, then add or change the *dns*= value to *none*:

```
$ sudo nano /etc/NetworkManager/NetworkManager.conf

[main]
dns=none
```

Enter your Dnsmasq server's IPv4 and IPv6 localhost addresses in */etc/resolv.conf* and your local domain, if you have one:

```
search sqr3l.nut
nameserver 127.0.0.1
nameserver ::1
```

Then reboot and configure your new Dnsmasq installation.

Discussion

NetworkManager and *systemd-resolved* are wonderful on client machines. On your Dnsmasq server, you must have control of */etc/resolv.conf*, and Dnsmasq should be the only stub resolver.

See Also

- *man 8 systemd-resolved.service*
- *man 8 networkmanager*

16.6 Configuring Dnsmasq for LAN DNS

Problem

You want to set up Dnmasq as your LAN DNS server.

Solution

Any hosts that you enter in */etc/hosts* need static IP addresses, and Dnsmasq will automatically enter them into DNS. At a minimum, enter your Dnsmasq server. The

following example includes the Dnsmasq server, a backup server, and an internal web server:

```
127.0.0.1 localhost
::1 localhost ip6-localhost ip6-loopback
192.168.43.81 dns-server
192.168.43.82 backups
192.168.43.83 https
```



Configure Static Hosts from DHCP

See Recipe 16.12 to learn how to manage static IP address assignments from DHCP, instead of */etc/hosts*.

Now it is time to configure Dnsmasq. Rename the default configuration file so you can start with a new empty file, and use the original as a reference:

```
$ sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf-old
$ sudo nano /etc/dnsmasq.conf
```

Copy the following configuration, replacing the second *listen-address* with your own server's IP address, and use your own domain name. The upstream name servers are OpenDNS, and you may use whatever upstream name servers you wish. Dnsmasq looks for */etc/resolv.conf* by default, but it doesn't hurt to be explicit:

```
# global options
resolv-file=/etc/resolv.conf
domain-needed
bogus-priv
expand-hosts
domain=sqr3l.nut
local=/sqr3l.nut/
listen-address=127.0.0.1
listen-address=192.168.43.81

# upstream name servers
server=208.67.222.222
server=208.67.220.220
```

Run Dnsmasq's syntax checker:

```
$ dnsmasq --test
dnsmasq: syntax check OK.
```

The syntax checker won't find configuration errors, but only typos. Start up Dnsmasq, and if there are errors it will not start. The following example shows a successful start:

```
$ sudo systemctl start dnsmasq.service
$ systemctl status dnsmasq.service
● dnsmasq.service - dnsmasq - A lightweight DHCP and caching DNS server
```

```
     Loaded: loaded (/lib/systemd/system/dnsmasq.service; enabled; vendor preset:
     enabled)
     Active: active (running) since Mon 2021-05-24 17:13:48 PDT; 1min 0s ago
    Process: 11023 ExecStartPre=/usr/sbin/dnsmasq --test (code=exited,
     status=0/SUCCESS)
    Process: 11024 ExecStart=/etc/init.d/dnsmasq systemd-exec (code=exited,
     status=0/SUCCESS)
    Process: 11033 ExecStartPost=/etc/init.d/dnsmasq systemd-start-resolvconf
     (code=exited, status=0/SUCCESS)
   Main PID: 11032 (dnsmasq)
      Tasks: 1 (limit: 18759)
     Memory: 2.5M
     CGroup: /system.slice/dnsmasq.service
             └─11032 /usr/sbin/dnsmasq -x /run/dnsmasq/dnsmasq.pid -u dnsmasq -7
              /etc/dnsmasq.d,.dpkg-dist,.dpkg-old,.dpkg-new --local->

May 24 17:13:48 dns-server systemd[1]: Starting dnsmasq - A lightweight DHCP and
 caching DNS server...
May 24 17:13:48 dns-server dnsmasq[11023]: dnsmasq: syntax check OK.
May 24 17:13:48 dns-server systemd[1]: Started dnsmasq - A lightweight DHCP and
 caching DNS server.
```

Run some tests on your Dnsmasq server with *nslookup* using your server's hostname and FQDN:

```
$ nslookup dns-server
Server:         127.0.0.1
Address:        127.0.0.1#53

Name:   dns-server
Address: 192.168.43.81

$ nslookup dns-server.sqr3l.nut
Server:         127.0.0.1
Address:        127.0.0.1#53

Name:   dns-server.sqr3l.nut
Address: 192.168.43.81

$ nslookup 192.168.43.81
18.43.168.192.in-addr.arpa      name = host1.sqr3l.nut.
```

Use the *ss* command to verify the listening ports. In the following example, the Recv-Q, Send-Q, and Peer Address:Port columns have been removed for clarity:

```
$ sudo ss -lp "sport = :domain"
Netid  State   Local Address:Port    Process
udp    UNCONN     127.0.0.1:domain   users:(("dnsmasq",pid=1531,fd=8))
udp    UNCONN  192.168.1.10:domain   users:(("dnsmasq",pid=1531,fd=6))
tcp    LISTEN     127.0.0.1:domain   users:(("dnsmasq",pid=1531,fd=9))
tcp    LISTEN  192.168.1.10:domain   users:(("dnsmasq",pid=1531,fd=7))
```

You should see your server address, localhost address, and only *dnsmasq* in the Process column. Add the *-r* option to see hostnames instead of IP addresses.

When all of these commands succeed, your configuration is correct.

Discussion

If Dnsmasq fails to start, run *journalctl -ru dnsmasq* to see why. (If your Dnsmasq logs are sent somewhere else, then look there; see Recipe 16.14.)

nslookup is in the *bindutils* package.

ss, socket statistics, is in the *iproute2* package.

If your *nslookup* commands fail, try restarting networking, and then restarting Dnsmasq. If they still fail, reboot. If this doesn't fix it, recheck all your configurations.

domain-needed prevents Dnsmasq from forwarding queries for your plain hostnames to upstream nameservers. If the name is not known from */etc/hosts* or DHCP, then a "not found" answer is returned. This keeps requests for your LAN addresses from leaking out into the world and possibly being answered incorrectly if your LAN domain is the same as a public domain name.

bogus-priv blocks bogus private reverse lookups. All reverse lookups for private IP ranges which are not found in */etc/hosts* or the DHCP leases file are answered with "no such domain" rather than being forwarded upstream.

expand-hosts automatically adds your private domain name to the plain hostnames in */etc/hosts*.

domain= is your local domain name.

local=/[domain]/ tells Dnsmasq to resolve queries for the local domain directly, and not forward them upstream.

See Also

- *man 5 hosts*
- Dnsmasq

16.7 Configuring firewalld to Allow DNS and DHCP

Problem

You need to open your Dnsmasq server's firewall to allow your LAN clients access to it.

Solution

Open TCP and UDP ports 53 for DNS, and UDP 67 for DHCP. If you are running *firewalld*, use this command:

```
$ sudo firewall-cmd --permanent --add-service=\{dns,dhcp\}
```

Discussion

One of the first things to check, when you have connectivity problems, is firewall settings.

See Also

- Chapter 14

16.8 Testing Your Dnsmasq Server from a Client Machine

Problem

You want to test your nice new Dnsmasq DNS server from a client computer.

Solution

Use the *dig* command from any host on your network to query any website, via the IP address for your Dnsmasq server:

```
$ dig @192.168.1.10 oreilly.com

; <<>> DiG 9.16.6 <<>> @192.168.1.10 oreilly.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29387
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;oreilly.com.                   IN      A

;; ANSWER SECTION:
oreilly.com.            240     IN      A       199.27.145.65
oreilly.com.            240     IN      A       199.27.145.64

;; Query time: 108 msec
;; SERVER: 192.168.1.10#53(192.168.1.10)
```

```
;; WHEN: Mon May 24 17:49:32 PDT 2021
;; MSG SIZE  rcvd: 72
```

That is a successful test, confirmed by "status: NOERROR" and the SERVER line showing the IP address of your Dnsmasq server.

Discussion

You can also test using your server's hostname and fully qualified domain name (FQDN):

```
$ dig @dns-server oreilly.com
$ dig @dns-server.sqr3l.nut oreilly.com
```

See Also

- *man 1 dig*

16.9 Managing DHCP with Dnsmasq

Problem

Your DNS is working, and now you want to set up DHCP.

Solution

No problem. Add these lines to your */etc/dnsmasq.conf* file to define a single pool of addresses, substituting your own desired addressing:

```
# DHCP range
dhcp-range=192.168.1.25,192.168.1.75,12h
dhcp-lease-max=25
```

Restart Dnsmasq:

```
$ sudo systemctl restart dnsmasq.service
```

Try getting an address on a LAN computer. First, make sure it is configured to get its IP address via DHCP:

```
$ nmcli con show --active
NAME    UUID                  TYPE     DEVICE
1net    de7c00e7-8e4d-45e6-acaf  ethernet eth0

$ nmcli con show 1net | grep ipv..method
ipv4.method:          auto
ipv6.method:          auto
```

auto confirms it is a DHCP client. (If it says *manual* then it is not.) Bring the interface down, then back up again:

```
$ sudo nmcli con down 1net
Connection '1net' successfully deactivated (D-Bus active path: /org/freedesktop/
NetworkManager/ActiveConnection/11

$ sudo nmcli con up 1net
Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkMan
ager/ActiveConnection/15)
```

Check your Dnsmasq server logs:

```
$ journalctl -ru dnsmasq
-- Logs begin at Sun 2021-02-28 14:35:01 PST, end at Mon 2021-05-31 17:36:04
 PDT. --
May 31 17:34:56 dns-server dnsmasq-dhcp[8080]: DHCPACK(eth0) 192.168.1.45
 9c:ef:d5:fe:01:7c client2
May 31 17:34:56 dns-server dnsmasq-dhcp[8080]: DHCPREQUEST(eth0) 192.168.1.45
9c:ef:d5:fe:01:7c
```

That shows a successful IP address assignment from *dns-server* to *client2*.

Discussion

You can use the NetworkManager panel applet instead of *nmcli*, or run the *nm-connection-editor* command to open NetworkManager's graphical configurator, then disconnect and connect with a mouse click (Figure 16-2).

Most Linux distributions use NetworkManager to control client DHCP. If yours does not, it probably uses *dhclient*. Look for a *dhclient.conf* configuration file, if this exists, then request a new lease with the *dhclient* command:

```
$ sudo dhclient -v
Internet Systems Consortium DHCP Client 4.3.6-P1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/eth0/9c:ef:d5:fe:01:7c
Sending on   LPF/eth0/9c:ef:d5:fe:01:7c
Sending on   Socket/fallback
DHCPREQUEST on eth0 to 255.255.255.255 port 67 (xid=0xec8923)
DHCPACK from 192.168.1.10 (xid=0xec8923)
bound to 192.168.1.27 -- renewal in 1415 seconds.
```

You can send, over DHCP, some of the information your client machines need to access network services. See Recipe 16.10 to learn more.

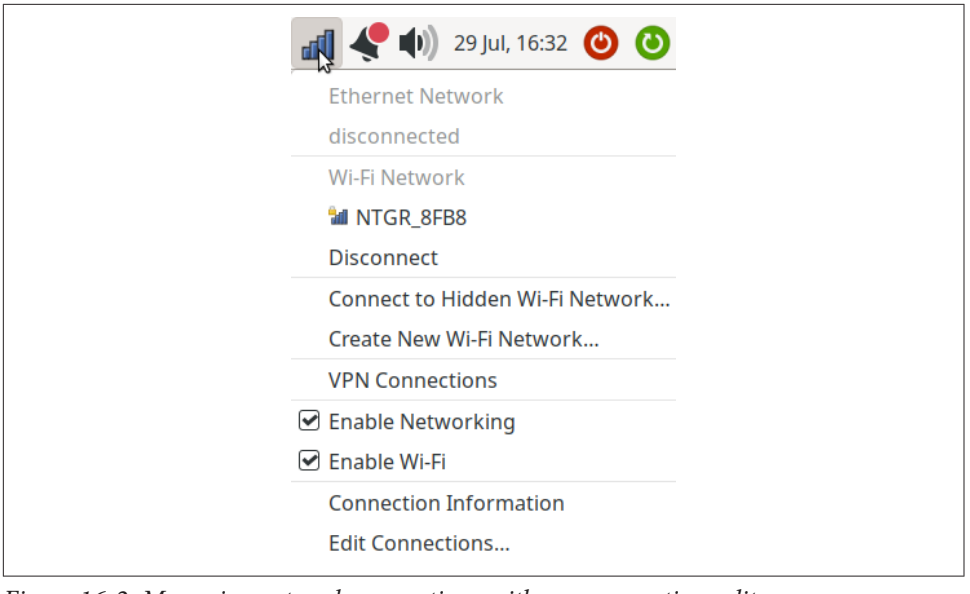


Figure 16-2. Managing network connections with nm-connection-editor

dhcp-range=192.168.1.25,192.168.10.75,24h defines a range of 50 available address leases, with a lease time of 24 hours. This range must not include your Dnsmasq server, nor any hosts with static IP addresses. Define the lease time in seconds, minutes, or hours. The default is one hour, and the minimum is two minutes. If you want leases that never expire, don't specify a lease time.

dhcp-lease-max=25 defines how many leases can be active at one time. You can have a large address pool available, and then limit the number of active leases.

See Also

- Recipe 16.10
- Dnsmasq
- *man 8 dhclient*

16.10 Advertising Important Services over DHCP

Problem

You want to advertise various servers to your LAN clients over DHCP.

Solution

Some services, like the default route to your internet gateway, DNS server, and NTP server, can be advertised to your LAN clients so that they automatically use them. The following examples show how to configure */etc/dnsmasq.conf* to advertise some services.

Set the default router:

```
dhcp-option=3,192.168.1.1
```

Advertise your DNS server:

```
dhcp-option=6,192.168.1.10
```

This example points the way to your local NTP server:

```
dhcp-option=42,192.168.1.11
```

How do you know which numbers to use? Use this command to list all of them:

```
$ dnsmasq --help dhcp
Known DHCP options:
  1 netmask
  2 time-offset
  3 router
  6 dns-server
  7 log-server
  9 lpr-server
  [...]
```

Discussion

dnsmasq --help dhcp displays the known DHCPv4 configuration options. See the Discussion in Recipe 16.11 for more information on the DHCPv4 configuration options.

See Also

- Dnsmasq

16.11 Creating DHCP Zones for Subnets

Problem

You have two subnets, and you want to configure Dnsmasq to apply different options to them, such as different default routers and servers.

Solution

Define your zones with whatever names you want to give them, like *zone1* and *zone2*, and set their address ranges:

```
dhcp-range=zone1,192.168.50.20,192.168.50.120
dhcp-range=zone2,192.168.60.20,192.168.60.50,24h
```

The two zones have different routers:

```
dhcp-option=zone1,3,192.168.50.1
dhcp-option=zone2,3,192.168.60.2
```

They use the same DNS server:

```
dhcp-option=zone1,6,192.168.1.10
dhcp-option=zone2,6,192.168.1.10
```

zone2 gets an NTP server:

```
dhcp-option=zone2,42,192.168.60.15
```

Discussion

Only a few of the DHCP options are useful. They are very old, and some are mysterious, for example:

> option default-url string;
>
> The format and meaning of this option is not described in any standards document, but is claimed to be in use by Apple Computer. It is not known what clients may reasonably do if supplied with this option. Use at your own risk.
>
> —man 5 DHCP options

Client support is inconsistent for many of them. The only ones I use are NTP, routers, and DNS servers.

See Also

- *man 5 dhcp*
- Dnsmasq

16.12 Assigning Static IP Addresses from DHCP

Problem

You want to centralize IP addressing as much as possible, including assigning static IP addresses.

Solution

Use the *dhcp-host* option in */etc/dnsmasq.conf*. Identify the client machine by its hostname, and assign an unused address from your LAN's address block. (It is not necessary to use the DHCP address range you defined with the *_dhcp-range=** option in */etc/dnsmasq.conf* for static addresses.) The following example assigns an address to *server2* in the 192.168.3.0/24 network:

```
dhcp-host=server2,192.168.3.45
```

Restart Dnsmasq, then the next time *server2* requests an address it will receive the address specified by the *dhcp-host=* option.

Use multiple *dhcp-host=* lines to configure multiple clients, one per line.

You may use the client's MAC address in place of the hostname.

Discussion

In general, centralizing administration chores saves time and headaches.

See Also

- Dnsmasq

16.13 Configuring DHCP Clients for Automatic DNS Entries

Problem

You want your DHCP clients to be entered into DNS automatically by Dnsmasq.

Solution

The only thing the clients have to do is send their hostnames to Dnsmasq's DHCP server, which is the default in most Linuxes.

Suppose a DHCP client on the local *sqr3l.nut* domain has the host name *client4*. *client4* starts up, and receives its IP address and other network information from Dnsmasq. Dnsmasq receives *client4*'s hostname and enters it into DNS. Now other hosts on the network can access *client4* and *client4.sqr3l.nut*.

There must not be any duplicate entries for *client4* in */etc/hosts*.

There are three different ways to check your DHCP client configuration: in *dhclient.conf*, NetworkManager's graphical configuration tool (*nm-connection-editor*), and with the *nmcli* command.

First check *dhclient*, which has been the default DHCP client on Linux for years. On most Linux systems its configuration file is */etc/dhcp/dhclient.conf*. Look for this line, which automatically finds the system's hostname and sends it to the DHCP server:

```
send host-name = gethostname();
```

Or a line like this, specifying the system's hostname:

```
send host-name = myhostname
```

If there is no *dhclient.conf* file, then NetworkManager is your DHCP client manager. You can check this in your graphical *nm-connection-editor* (Figure 16-3).

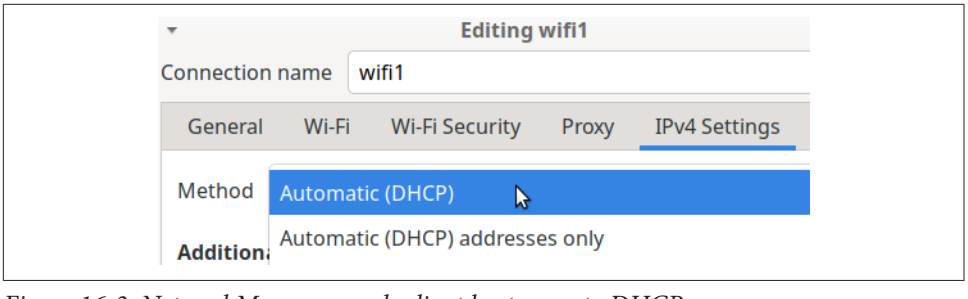


Figure 16-3. NetworkManager sends client hostname to DHCP server

When the connection method is "Automatic (DHCP)," NetworkManager sends the hostname to the DHCP server. "Automatic (addresses only)" does not send the hostname to the DHCP server, but only provides DNS to the client.

You may also use the *nmcli* command. First, find your active network connection:

```
$ nmcli connection show --active
NAME    UUID                                  TYPE      DEVICE
wifi1   3e348c97-4c5f-4bbf-967e-7624f3e1e4f0  wifi      wlan1
```

Then verify that it sends the hostname to your DHCP server. The following example confirms that it does:

```
$ nmcli connection show wifi1 | grep send-hostname
ipv4.dhcp-send-hostname:                yes
ipv6.dhcp-send-hostname:                yes
```

If it says *no*, run the following commands to set it to *yes*. After that, reload the configuration:

```
$ sudo nmcli con mod wifi1 ipv4.dhcp-send-hostname yes
$ sudo nmcli con mod wifi1 ipv6.dhcp-send-hostname yes
$ sudo nmcli con reload
```

Discussion

If you prefer a graphical tool to manage NetworkManager, it's best to use NetworkManager's graphical configuration utility, *nm-connection-editor*, rather than a different graphical tool, such as the network module in the GNOME control panel. The *nm-connection-editor* offers the most complete configuration options, and it is the same on all Linux distros.

See Also

- *man 1 nmcli*
- *man 1 nmcli-examples*
- *man 5 nm-settings*

16.14 Managing Dnsmasq Logging

Problem

Dnsmasq has the option to send its messages to a file of your choice using the legacy *syslog* daemon, rather than to *journalctl*, and you want to know which is the best option.

Solution

It does not matter which one you use: the same information is logged either way. The default behavior is to log to the systemd journal.

It can be convenient to isolate Dnsmasq logs in their own directory, such as */var/log/dnsmasq/dnsmasq.log*. Use the *log-facility=* option in */etc/dnsmasq.conf* to specify the log file you want to use, then restart Dnsmasq. The file must already exist or Dnsmasq will not start.

Your logfile will grow very large unless you set up log rotation. The following example configuration, */etc/logrotate.d/dnsmasq*, sets up a simple weekly rotation:

```
/var/log/dnsmasq/dnsmasq.log {
    missingok
    compress
    notifempty
    rotate 4
    weekly
    create
    }
```

Test it with the *logrotate* command:

```
$ sudo /etc/logrotate.conf --debug
[...]
rotating pattern: /var/log/dnsmasq/dnssmasq.log  weekly (4 rotations)
empty log files are not rotated, old logs are removed
switching euid to 0 and egid to 4
considering log /var/log/dnsmasq/dnssmasq.log
Creating new state
  Now: 2021-06-01 13:08
  Last rotated at 2021-06-01 13:00
  log does not need rotating (log has been already rotated)
switching euid to 0 and egid to 0
[...]
```

This shows no errors and it is working correctly.

Discussion

systemd supports both *journalctl* and the *syslog* daemon. They will probably exist together for a long time, so you can set up logging in whatever way you prefer.

See Also

- *man 8 rsyslog*
- Dnsmasq
- *man 1 journalctl*
- Chapter 20

16.15 Configuring Wildcard Domains

Problem

You want to create a wildcard domain in Dnsmasq, so that requests for the domain's subdomains resolve without manually adding the subdomains to your DNS.

Solution

Use the *address* option in */etc/dnsmasq.conf* to create the top-level domain (TLD):

```
address=/wildcard.net/192.168.1.35
```

Restart Dnsmasq, then run *nslookup* to test:

```
$ sudo systemctl restart dnsmasq.service
$ nslookup foo.wildcard.net
Server:         127.0.0.1
```

```
Address:        127.0.0.1#53

Name:   foo.wildcard.net
Address: 192.168.1.35
```

foo.wildcard.net resolves, showing that it works.

Discussion

Use DNS wildcards carefully. Wildcards are useful when you're doing development work on complex services such as Kubernetes. Make sure to use address ranges that are different from the ranges on your LAN's name server, and are available only to LAN clients.

See Also

- Dnsmasq

CHAPTER 17

Keeping Time with ntpd, chrony, and timesyncd

Keeping accurate time on your computer, and on all hosts on your network, is easy and automatic with NTP, the Network Time Protocol. NTP is implemented on Linux with *ntpd*, the NTP daemon, *chrony*, the modern replacement for *ntpd*, and systemd's *timesyncd*. That is right, friends, there are three (at least), count them, three ways to automatically manage time on your Linux computer.

ntpd and *chrony* can also function as LAN time servers, while *timesyncd* is a simpler lightweight client with no server functions. *ntpd* and *chrony* are full NTP implementations, while *timesyncd* uses SNTP, the Simple Network Time Protocol.

Most Linux distributions provide a default configuration that points to time servers that they maintain. These servers have names like *2.fedora.pool.ntp.org* and *0.ubuntu.pool.ntp.org*. You don't have to do anything, except be sure to not disable this during installation. In this chapter you will learn how to check your current settings, how to change them, and how to set up a LAN time server.

There is a worldwide network of time servers that are free for everyone to use, and they are organized into *strata*, starting at stratum 0. Stratum 0 is the source for all timekeeping, a network of atomic clocks, radio receivers tuned to atomic clocks, and GPS receivers using signals broadcast by GPS satellites.

Next in line is stratum 1, where the primary time servers reside. The primary time servers in stratum 1 are directly connected to the sources in stratum 0.

Stratum 2 contains thousands of public servers, and they sync with stratum 1. It is good etiquette to connect to stratum 2 servers to prevent the stratum 1 servers from being overwhelmed and to not use the stratum 1 servers without a good reason.

The hierarchy continues on down the line, for example there are stratum 4, 5, and 6 public servers, and private LAN servers that sync with them. It's not really that orderly; you can designate your private LAN NTP server as stratum 10, and it doesn't have to connect to stratum 9 servers, but any server it can reach. You don't have to worry about selecting the correct servers because you will use *pool* servers, which are clusters of NTP servers, rather than individual servers.

When you dig into timekeeping on computers, it becomes confusing and overwhelming, or perhaps fascinating, depending on how nerdy you want to get. Visit the NTP Pool Project and NTP: The Network Time Protocol to learn the nerdy stuff and how to run your own public time server.

There are at least two timekeepers on your Linux system. One is the hardware clock on your motherboard, which is also called the real-time clock (RTC). The other is system time, managed by your Linux kernel. The RTC always has power, even when your machine is turned off, from a battery or capacitor on the motherboard. When your Linux computer starts up, your selected NTP client gets its time from the RTC. Then, after the network is available, it corrects the time according to its upstream time server.

Your RTC time is set in your BIOS/UEFI, and with some of the commands you will learn about in this chapter. It should always be set to UTC, Coordinated Universal Time, and then the Linux kernel calculates the time for your time zone from the UTC. UTC is similar to Greenwich Mean Time (GMT), though they are not the same. UTC is a time standard and GMT is a time zone. Neither UTC nor GMT change for daylight saving time (DST).

Time zone data comes from IETF.org Timezones. This is a moving target as countries change their DST dates, opt out of DST, and opt back in. Most Linuxes store this information in */usr/share/zoneinfo/*. The Internet Engineering Task Force (IETF) tracks these changes and makes their databases freely available.

17.1 Finding Which NTP Client Is on Your Linux System

Problem

You read the chapter introduction, and now you know that time synchronization on Linux is managed by *ntpd*, *chrony*, or *timesyncd*, and you need to know which one your Linux system uses.

Solution

Use the *ps* command to see if any of the three time synchronization daemons, *ntpd*, *chronyd*, or *timesyncd* are running on your system:

```
$ ps ax|grep -w ntp
$ ps ax|grep -w chrony
$ ps ax|grep -w timesyncd
```

If any of these are running, skip ahead to the relevant recipes in this chapter to learn how to manage your time daemon.

If none of these are running, see if your system is using *timedatectl*, which is part of systemd:

```
$ timedatectl status
                      Local time: Sun 2020-10-04 10:59:48 PDT
                  Universal time: Sun 2020-10-04 17:59:48 UTC
                        RTC time: Sun 2020-10-04 17:59:48
                       Time zone: America/Los_Angeles (PDT, -0700)
       System clock synchronized: no
systemd-timesyncd.service active: no
                 RTC in local TZ: no
```

This output shows that *timedatectl* is running without any time daemons, indicated by *systemd-timesyncd.service active: no*. Double-check by querying the status of *systemd-timesyncd*:

```
$ systemctl status systemd-timesyncd
● systemd-timesyncd.service - Network Time Synchronization
   Loaded: loaded (/lib/systemd/system/systemd-timesyncd.service; disabled;
vendor preset: enabled)
   Active: inactive (dead)
     Docs: man:systemd-timesyncd.service(8)
```

This shows that *systemd-timesyncd* is not running, which means there is no time synchronization on your system, and it is getting its time from your system's real-time clock (RTC). In this case you need to set up *ntpd*, *chrony*, or *timesyncd*.

Discussion

Some Linux distributions do not use systemd; see Recipe 4.1 to learn how to know if your Linux has it. If you are running a Linux system without systemd, your NTP choices are *ntpd* or *chrony*.

If you find both *ntpd* and *chrony* running on the same system, get rid of *ntpd*, as *chrony* is newer, faster, and more reliable. Having both will create conflicts.

The output of *timedatectl* has a lot of useful information. The example shows that the RTC is correctly set to the Coordinated Universal Time (UTC) protocol, and that the system time zone is Pacific Daylight Time, PDT. *systemd-timesyncd.service* is not running, and the system has not been synchronized.

See Also

- timedatectl: Control the system time and date
- *man 1 ps*

17.2 Using timesyncd for Simple Time Synchronization

Problem

You want to know how to set up the simplest NTP client to keep the correct time on your computer.

Solution

Enable synchronization with a public NTP server using the *systemd-timesyncd* daemon, which requires systemd. Check the status of *systemd-timesyncd*:

```
$ systemctl status systemd-timesyncd
● systemd-timesyncd.service - Network Time Synchronization
     Loaded: loaded (/usr/lib/systemd/system/systemd-timesyncd.service;
       disabled; vendor preset: enabled)
     Active: inactive (dead)
       Docs: man:systemd-timesyncd.service(8)
```

Enable it with *timedatectl*, and verify that *systemd-timesyncd* started:

```
$ timedatectl set-ntp true
$ systemctl status systemd-timesyncd
● systemd-timesyncd.service - Network Time Synchronization
   Loaded: loaded (/lib/systemd/system/systemd-timesyncd.service; enabled;
vendor preset: enabled)
   Active: active (running) since Sun 2020-10-04 18:17:51 PDT; 16min ago
     Docs: man:systemd-timesyncd.service(8)
 Main PID: 3990 (systemd-timesyn)
   Status: "Synchronized to time server 91.189.89.198:123 (ntp.ubuntu.com)."
    Tasks: 2 (limit: 4915)
   CGroup: /system.slice/systemd-timesyncd.service
           └─3990 /lib/systemd/systemd-timesyncd

Oct 04 18:17:51 pc systemd[1]: Starting Network Time Synchronization...
Oct 04 18:17:51 pc systemd[1]: Started Network Time Synchronization.
Oct 04 18:33:01 pc systemd-timesyncd[3990]: Synchronized to time server
91.189.89.198:123 (ntp.ubuntu.com).
```

If *systemd-timesyncd* did not start, start it:

```
$ sudo systemctl start systemd-timesyncd
```

Now see what *timedatectl* reports:

```
$ timedatectl status
                      Local time: Sun 2020-10-04 18:35:56 PDT
                  Universal time: Mon 2020-10-05 01:35:56 UTC
                        RTC time: Mon 2020-10-05 01:35:56
                       Time zone: America/Los_Angeles (PDT, -0700)
       System clock synchronized: yes
systemd-timesyncd.service active: yes
                 RTC in local TZ: no
```

Everything looks correct. Your system is synchronized, all the times are correct, and the *systemd-timesyncd.service* is active.

It is a good practice to configure multiple public time servers for redundancy. Edit */etc/systemd/timesyncd.conf* to add more NTP servers by uncommenting the `NTP` line and entering a space-delimited list of public server pools:

```
[Time]
NTP=0.north-america.pool.ntp.org 1.north-america.pool.ntp.org
2.north-america.pool.ntp.org
#FallbackNTP=ntp.ubuntu.com
#RootDistanceMaxSec=5
#PollIntervalMinSec=32
#PollIntervalMaxSec=2048
```

Discussion

In your original */etc/systemd/timesyncd.conf* file, the commented options document the default configuration.

The pool servers are highly reliable because they are multiple servers in a single pool, rather than individual servers. For best peformance use the pool servers for your region, either the continental pools or the country pools, which you will find by clicking on the continental pool links.

Your Linux distribution may configure multiple server pools of their own, such as:

```
0.opensuse.pool.ntp.org 1.opensuse.pool.ntp.org 2.opensuse.pool.ntp.org
```

This is good, and you don't need to change it, but a more diverse configuration is usually more reliable.

See Also

- Chapter 4
- NTP Pool Project
- *man 5 timesyncd.conf*

17.3 Setting Time Manually with timedatectl

Problem

You want to set your system and RTC time manually.

Solution

Use *timedatectl*. It sets the date, system time, and RTC time with a single command:

```
$ timedatectl set-time "2020-10-04 19:30:00"
Failed to set time: Automatic time synchronization is enabled
```

You cannot do this when *systemd-timesyncd* is running, so you must stop it:

```
$ sudo systemctl stop systemd-timesyncd
```

Then enter your new settings in the format shown in the example, YYYY-MM-DD HH:MM:SS, and verify that it worked:

```
$ timedatectl set-ntp false
$ timedatectl set-time "2020-10-04 19:30:00"
$ timedatectl status
                      Local time: Sun 2020-10-04 19:30:06 PDT
                  Universal time: Mon 2020-10-05 02:30:06 UTC
                        RTC time: Mon 2020-10-05 02:30:06
                       Time zone: America/Los_Angeles (PDT, -0700)
       System clock synchronized: no
systemd-timesyncd.service active: no
                 RTC in local TZ: no
```

If you restart *systemd-timesyncd*, it will override your manual settings.

Discussion

timedatectl has a small set of commands. If you are used to the *date* command for setting the time, and other time and date operations, *timedatectl* may seem lightweight. It is simple by design, and you still have *date* and its numerous options for complex tasks.

See Also

- *man 5 timesyncd.conf*

17.4 Using chrony for Your NTP Client

Problem

Your want a fully featured NTP client/server, and you want to know how to set up *chrony* as your NTP client.

Solution

First, check if *ntpd* is installed. If it is, remove it. If you have *systemd-timesyncd*, disable it:

```
$ sudo systemctl disable systemd-timesyncd
$ sudo systemctl stop systemd-timesyncd
```

Then install *chrony*. The package name on most Linuxes is *chrony*. After installation use the *chronyc* command to check its status:

```
$ chronyc activity
200 OK
8 sources online
0 sources offline
0 sources doing burst (return to online)
0 sources doing burst (return to offline)
0 sources with unknown address
```

Success! It is already working, as *8 sources online* tells you. (If it did not start, see the Discussion.) Find your *chrony.conf*, either */etc/chrony.conf* (Fedora) or */etc/chrony/chrony.conf* (Ubuntu), and take a look at the settings. There is not much you need to change, if anything, to use it as a client. Check the NTP server list, where you will see either *server* options or *pool*. The following example on an Ubuntu system includes the default Ubuntu NTP server pools and a local LAN server:

```
pool 0.ubuntu.pool.ntp.org iburst
pool 1.ubuntu.pool.ntp.org iburst
pool 1.ubuntu.pool.ntp.org iburst
server ntp.domain.lan iburst prefer
```

You could replace some of the Ubuntu server pools with some public server pools to improve reliability with a more diverse set of pools:

```
pool 0.ubuntu.pool.ntp.org iburst
pool 1.ubuntu.pool.ntp.org iburst
pool 0.north-america.pool.ntp.org iburst
pool 1.north-america.pool.ntp.org iburst
server ntp.domain.lan iburst prefer
```

Restart *chronyd* after changing the configuration file.

Discussion

iburst means synchronize quickly after network interruptions, and *prefer* means always use this server, unless it is not available.

That really is all you need to do for a client setup. *chrony* is a full NTP implementation and has many options; see *man 5 chrony.conf* for a complete description.

Manage *chronyd* just like any other service using the following commands:

- *systemctl status chrony*
- *sudo systemctl stop chrony*
- *sudo systemctl start chrony*
- *sudo systemctl restart chrony*

Chrony has several advantages over *ntpd*. The main advantages as a client are better handling of interrupted network connections and faster resyncing when the connection is restored.

See Also

- chrony
- *man 5 chrony.conf*
- *man 1 chronyc*

17.5 Using chrony as a LAN Time Server

Problem

You want to set up *chrony* as your LAN time server.

Solution

Just like in Recipe 17.4, disable *systemd-timesyncd* and remove *ntpd* if it is on your system. Then install the *chrony* package.

Find the configuration file, either */etc/chrony.conf* (Fedora, openSUSE) or */etc/chrony/chrony.conf* (Ubuntu). The following example is a basic configuration:

```
pool 0.north-america.pool.ntp.org iburst
pool 1.north-america.pool.ntp.org iburst
pool 2.north-america.pool.ntp.org iburst

local stratum 10
```

```
allow 192.168.0.0/16
allow 2001:db8::/56

driftfile /var/lib/chrony/chrony.drift
maxupdateskew 100.0
rtcsync
logdir /var/log/chrony
log measurements statistics tracking
leapsectz right/UTC
makestep 1 3
```

Then your clients need your server's name added to their *chrony.conf* files:

```
server ntp.domain.lan iburst prefer
```

The *prefer* option means to always use this server as long as it is available. One of the reasons to keep a local time server is to put less of a load on the public time servers. With the *prefer* option you can configure some public servers as backups, in case your local server becomes unavailable, and not worry about burdening them, like this:

```
server ntp.domain.lan iburst prefer
pool 1.north-america.pool.ntp.org iburst
pool 2.north-america.pool.ntp.org iburst
```

Discussion

local stratum 10 configures *chrony* to continue acting as your local NTP server even when your internet connection is interrupted, and *stratum 10* puts your server safely down the strata hierarchy, so that it is lower than any external NTP servers you are using. Allowed values are 1–15. (Do please use a number other than 10, in case this recipe makes *stratum 10* wildly popular.)

The *allow* options define the networks that are allowed to use your NTP server.

rtcsync tells *chrony* to keep your RTC synchronized with system time.

log enables logging and defines the events you want logged.

You may look up the other options in *man 5 chrony.conf*, or in your default *chrony.conf*, which is usually well commented.

See Also

- chrony
- *man 5 chrony.conf*
- *man 1 chronyc*

17.6 Viewing chrony Statistics

Problem

You want to call up some real-time *chrony* activity and statistics, such as upstream NTP servers, offsets, skew, which server you are currently synced with, and other information.

Solution

Use the *chronyc* command. The *tracking* subcommand shows how much correction has been applied, the RTC time, skew, and other information:

```
$ chronyc tracking
Reference ID    : A29FC87B (time.cloudflare.com)
Stratum         : 4
Ref time (UTC)  : Tue Oct 06 02:20:23 2020
System time     : 0.002051390 seconds fast of NTP time
Last offset     : +0.002320110 seconds
RMS offset      : 0.017948814 seconds
Frequency       : 28.890 ppm fast
Residual freq   : +0.252 ppm
Skew            : 1.250 ppm
Root delay      : 0.069674924 seconds
Root dispersion : 0.003726898 seconds
Update interval : 838.2 seconds
Leap status     : Normal
```

List your current source servers:

```
$ chronyc sources
chronyc sources
210 Number of sources = 19
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^- golem.canonical.com           2   9     0   37m    +55ms[  +58ms] +/-  209ms
^- alphyn.canonical.com          2   9     0   34m    +23ms[  +25ms] +/-  158ms
^- pugot.canonical.com           2   9     0   44m    +92ms[  +80ms] +/-  229ms
^- chilipepper.canonical.com     2   9    11    31    +48ms[  +48ms] +/-  181ms
[...]
```

List your current source servers, with descriptions:

```
$ chronyc sources -v
210 Number of sources = 19

  .-- Source mode  '^' = server, '=' = peer, '#' = local clock.
 / .- Source state '*' = current synced, '+' = combined , '-' = not combined,
| /   '?' = unreachable, 'x' = time may be in error, '~' = time too variable.
||                                               .- xxxx [ yyyy ] +/- zzzz
||      Reachability register (octal) -.         |  xxxx = adjusted offset,
||      Log2(Polling interval) --.      |        |  yyyy = measured offset,
```

```
||                                 \     |          |  zzzz = estimated error.
||                                  |    |           \
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^- golem.canonical.com           2   9     0   46m    +67ms[  +58ms] +/-  209ms
^- alphyn.canonical.com          2   9     0   44m    +35ms[  +25ms] +/-  158ms
^* pugot.canonical.com           2   9     1   54m   +104ms[  +80ms] +/-  229ms
^- chilipepper.canonical.com     2   9    11   587    +60ms[  +48ms] +/-  181ms
^- ntp.wdc1.us.leaseweb.net      2   7     4   327    +26ms[  +15ms] +/-  198ms
^- 216.126.233.109               2   9     1   459   +106ms[  +95ms] +/-  171ms
^- 157.245.170.163               3   9     1   476  +1191us[  -10ms] +/-  145ms
```

The asterisk shows which server your system is currently synchronizing with.

Discussion

chrony adjusts for network delays and latency, intermittent connections, and sleep and hibernate modes on client machines. The *chrony* clock never stops, and it keeps your network synchronized even when external name servers are unavailable.

See Also

- *man 1 chronyc*
- chrony.tuxfamily.org

17.7 Using ntpd for Your NTP Client

Problem

Yes, you know all about *chrony* and *timesyncd*, and how good they are, but you still want to use *ntpd* as your NTP client.

Solution

No problem, because *ntpd* is actively maintained and does the job just fine. First, make sure that *ntpd* is the only NTP client on your system (see Recipe 17.1). On most Linux distributions, look for the *ntp* package to install.

On most Linux distributions *ntpd* comes with a useful configuration, and starts after installation. Check with the *ps* command:

```
$ ps ax | grep -w ntpd
3754 ?        Ssl    0:00 /usr/sbin/ntpd -u ntp:ntp -g
```

If it does not start automatically, start it:

```
$ systemctl start ntpd
```

While *ntpd* is running, take a look at your configuration file, usually */etc/ntp.conf*. The default configuration for your Linux distribution should work fine for you without changes. If your network has its own LAN server, the following configuration sets the local server as the primary and one Fedora Linux server pool as a fallback:

```
server ntp.domain.lan iburst prefer
pool 2.fedora.pool.ntp.org iburst
```

You may keep the default configuration, which works fine for most situations. It is common for Linux distributions to maintain their own NTP server pools and provide these in the default configuration. If you wish to replace these, or add some external public servers, see continental pools for a list of the continental NTP server pools, or use your country pools, which you will find by clicking on the continental pool links.

When you change */etc/ntp.conf*, restart *ntpd*:

```
$ systemctl restart ntpd

$ sudo /etc/init.d/ntp restart
```

Check that it is working with *ntpq*. The asterisk shows that the machine is syncing with the LAN NTP server:

```
$ ntpq -p

     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
 2.fedora.pool.n .POOL.          16 p    -   64    0    0.000   +0.000   0.000
*ntp.domain.lan. 172.16.16.3      2 u   34  256  203   80.324  -49.772  54.508
+138.68.46.177 ( 80.153.195.191   2 u   92  256  123   90.932  -15.534  39.947
+vps6.ctyme.com  216.218.254.202  2 u  453  256   46   69.927  -29.296  84.811
+ec2-3-217-79-24 132.163.97.6     2 u  426  256  202  165.888  -51.442  93.224
```

Discussion

iburst tells *ntpd* to synchronize quickly at system startup.

prefer means use this server, and use the others only when it becomes unavailable.

See Also

- *man 5 ntp.conf*
- *man 8 ntpd*
- *man 8 ntpq*
- NTP Documentation

17.8 Using ntpd for Your NTP Server

Problem

You want to know how to run an *ntpd* server for your LAN.

Solution

Using *ntpd* for your LAN time server is similar to using it as an NTP client. The configuration is almost the same, with the addition of some access controls. The following example is a complete */etc/ntp.conf* configuration:

```
driftfile /var/lib/ntp/drift

restrict default nomodify notrap nopeer noquery
restrict -6 default nomodify notrap nopeer noquery
restrict 127.0.0.1
restrict ::1

pool 0.north-america.pool.ntp.org
pool 1.north-america.pool.ntp.org
pool 2.north-america.pool.ntp.org

leapfile /usr/share/zoneinfo/leap-seconds.list

statistics clockstats loopstats peerstats
filegen loopstats file loopstats type day enable
filegen peerstats file peerstats type day enable
filegen clockstats file clockstats type day enable
statsdir /var/log/ntpstats/
```

Discussion

The driftfile is where *ntpd* tracks the time drift caused by fluctuations in the oscillating frequency of the quartz crystal on your motherboard. You have the following options:

- *restrict default* denies all, permits only what is explicitly allowed, and sets defaults.
- *nomodify* does not allow other time servers to make any changes on your system. Queries are allowed.
- *notrap* disables remote logging.
- *nopeer* does not allow peering. Peer servers synchronize with each other, so the only servers allowed to supply time service are specified by the *server* or *pool* directives.

- *noquery* disallows remote queries and remote logging.
- *restrict 127.0.0.1* and *restrict ::1* mean trust localhost.

The *statistics* section logs your selected statistics into */var/log/ntpstats/*. This is not necessary, but it could be interesting for tracking which upstream NTP servers have the best performance.

See Also

- *man 5 ntp.conf*
- *man 8 ntpd*
- *man 8 ntpq*
- NTP Documentation

17.9 Managing Time Zones with timedatectl

Problem

You want to list all time zones, see your current time zone, and change time zones.

Solution

Use *timedatectl*. View your current time zone:

```
$ timedatectl | grep -i "time zone"
     Time zone: America/Los_Angeles (PDT, -0700)
```

List all time zones:

```
$ timedatectl list-timezones
Africa/Abidjan
Africa/Accra
Africa/Addis_Ababa
Africa/Algiers
[...]
```

The list is over 400 lines long. Use the *grep* command when you know what you are looking for. The list names major cities, for example, Berlin:

```
$ timedatectl list-timezones | grep -i berlin
Europe/Berlin
```

Set this as your time zone:

```
$ sudo timedatectl set-timezone Europe/Berlin
```

The change takes effect immediately. Run *timedatectl* again to verify.

Discussion

When you work with people in different time zones, use UTC to coordinate meeting times. There are several online time zone converters, such as Time Zone Converter.

You must use the region/city form, as displayed by *timedatectl list-timezones*, to set your time zone. This is defined by the ISO 8601 standard, which defines an unambiguous standard for expressing time zones, time, and dates. The standard uses a "descending notation," which is the largest values to the smallest values, in sequence. For example, for time zones the order is continent/country/city. The US habit of writing the date as year-day-month does not conform to this standard. Year-month-day is the standard, with a 4-digit year, YYYY-MM-DD. Time is HH:MM:SS, in the 24-hour clock format.

The officially published ISO 8601 standard costs money, but a bit of web searching should find the information for free.

See Also

- *man 1 timedatectl*

17.10 Managing Time Zones Without timedatectl

Problem

Your Linux system does not have systemd, and you need to know what commands to use to manage time zones.

Solution

Use the *date* command to see your current time zone:

```
$ date
Wed Oct  7 08:32:40 PDT 2020
```

Or see what */etc/localtime* is linked to:

```
$ ls -l /etc/localtime
lrwxrwxrwx 1 root root 41 Oct  7 08:06 /etc/localtime ->
../usr/share/zoneinfo/America/Los_Angeles
```

The */usr/share/zoneinfo* directory contains all time zones:

```
$ ls /usr/share/zoneinfo
total 324
drwxr-xr-x  2 root root  4096 May 21 23:02 Africa
drwxr-xr-x  6 root root 20480 May 21 23:02 America
drwxr-xr-x  2 root root  4096 May 21 23:02 Antarctica
```

```
drwxr-xr-x  2 root root  4096 May 21 23:02 Arctic
[...]
```

Look in the subdirectories to find a city close to yours, for example, Madrid:

```
$ ls /usr/share/zoneinfo/Europe
[...]
-rw-r--r-- 1 root root 2637 May  7 17:01 Madrid
-rw-r--r-- 1 root root 2629 May  7 17:01 Malta
lrwxrwxrwx 1 root root    8 May  7 17:01 Mariehamn
-rw-r--r-- 1 root root 1370 May  7 17:01 Minsk
[...]
```

Change your time zone by changing the link to */etc/localtime*:

```
$ sudo ln -sf /usr/share/zoneinfo/Europe/Madrid/etc/localtime
```

The change takes effect immediately.

Discussion

This cool command lists all time zones alphabetically:

```
$ php -r 'print_r(timezone_identifiers_list());'
Array
(
    [0] => Africa/Abidjan
    [1] => Africa/Accra
    [2] => Africa/Addis_Ababa
    [3] => Africa/Algiers
    [4] => Africa/Asmara
[...]
```

The *php* command is in the *php-cli* package.

Your graphical desktop should have a nice graphical utility for managing time, date, and time zones. If a clock is displayed on your desktop, try right-clicking it to bring up a properties or settings panel.

See Also

- *man 1 date*
- *man 1 ln*

CHAPTER 18

Building an Internet Firewall/Router on Raspberry Pi

The Raspberry Pi (RPi) makes a great internet firewall/router for small networks, and it does not cost a lot of money. You can use any Raspberry Pi, but I recommend the Raspberry Pi 4B because it is more powerful than the older Pis and is the first Pi with a dedicated gigabit Ethernet port.

Overview

In this chapter you will learn how to install Raspberry Pi OS, connect your Pi to a computer monitor or TV, use the Raspberry Pi OS recovery mode, run your Pi headless, add a second Ethernet port, share an internet connection, and use the Pi for LAN name services.



The examples in this chapter are all on a Raspberry Pi 4 Model B, using the Raspberry Pi OS. Raspberry Pi OS used to be called Raspbian. Underneath it is Debian Linux, so if you are accustomed to Debian, Ubuntu, Mint, or any other Debian variant, it's the same Linux.

Pros and Cons of a Raspberry Pi Firewall/Router

The Raspberry Pi is a general-purpose computer, not a specialized firewall/router. It has WiFi, Ethernet, and Bluetooth, and it runs Linux. In comparison, a common choice for small networks is the small combination firewall/router/wireless access point/ Ethernet switch, like the Linksys AC1900 or the TP-Link Archer AX20. These have WiFi, gigabit Ethernet, multiple antennas, and support for "smart" services like Alexa and smartphone administration apps.

The drawback to these devices is their inflexibility, especially limited storage and operating system support. If you want to replace the vendor software, you must use specialized router distributions like OpenWRT, DD-WRT, pfSense, or OPNsense, which are all excellent, but it is not easy, and you have to find supported devices.

These are some of the advantages Raspberry Pi has over the all-in-one boxes:

- Flexibility, just like any general-purpose Linux computer
- More memory and storage
- Connects to a computer monitor or TV, keyboard, and mouse
- Runs a number of Linux distributions, so you can use your existing knowledge and not have to learn some weird new interface or command set
- Runs a number of *bsd operating systems, Windows 10, Android, Chromium, and others
- Tethers to a mobile hotspot
- 64-bit support

You can run the Raspberry Pi with a graphical desktop, with no graphical desktop, and headless via SSH, just like any Linux system.

Downsides of Raspberry Pi:

- Cannot function as an Ethernet switch, like the all-in-one devices
- WiFi is not as strong as all-in-one devices
- Raspberry Pi models 3 and older have poor Ethernet performance because the Ethernet port shares the USB bus; RPi 4 cures this with a dedicated gigabit Ethernet port

There are a number of customized Linux operating systems for Raspberry Pi. The official operating system is Raspberry Pi OS, which is based on Debian Linux. SUSE, Ubuntu, Fedora, Arch Linux ARM, and MX Linux have Raspberry Pi variants. There are also specialized media server and gaming distros. In this chapter we'll stick with Raspberry Pi OS because it is optimized for the Pi, and you get a full graphical desktop with decent performance even on the older RPi models. Raspberry Pi OS is just like any other Linux, so anything you can do on a big Linux machine you can do on Raspberry Pi.

Hardware Architecture

The RPi is powered by a Broadcom system-on-a-chip (SoC). The CPU, GPU, and I/O are all on a single chip. This is especially impressive on the Raspberry Pi 4 Model B, which supports running two screens at the same time. It handles high-definition movies without a hiccup.

The Broadcom SoC is an ARM chip, rather than the x86_64 architecture that dominates the PC market. ARM processors are less complex, using a reduced instruction set (RISC). x86_64 processors are CISC, complex instruction set computers. x86_64 processors work a lot harder, are considerably more complex, and consume more power.

Rasberry Pi Banquet of Products

Every Raspberry Pi model ever made is still available, including updated releases of the Raspberry Pi 1, models A and B. The A models are the lower-cost versions of every release, and the B models cost a little more for more features.

You have many other Pis to choose from, such as:

- Raspberry Pi Zero, the smallest Pi for $5
- Raspberry Pi Zero W includes WiFi, $10
- Raspberry Pi 400 Personal Computer Kit, a complete system integrated into a compact keyboard (all you need to add is a display), $100

RPi prices have been stable since the first RPi was released, though of course this could change.

There is a giant thundering herd of accessories: cases, touchscreens, heat sinks, fans, breakout boards, hats (expansion boards), all manner of cables and adapters, motors, cameras, audio boards, little wireless keyboards with touchpads, gaming emulators, RGB matrices, powered USB hubs, real-time clocks, tiny displays…it is one of the best playgrounds ever.

History and Purpose

The Raspberry Pi is a real phenomenon. The original intent of its creator, Eben Upton, was to produce a small and inexpensive computer to encourage young students to study computing, especially students who could not afford to buy a PC. The first Raspberry Pi, Version 1 Model B, cost about $35. Add a keyboard and mouse, connect to a TV or monitor, and for less than a tenth the price of a PC you had a working Linux computer with audio, video, Ethernet, and USB. The open hardware design, combined with open source software, encourages hacking and learning.



The Broadcom chipsets that power the Pi are not open. This has been a point of contention since the first Pi was released. The schematics are open, available on RaspberryPi.org, and the operating system and BIOS are open source. In my moderately humble opinion, a completely open source platform is preferable and would be great, and having something that works and is available now is better than not having something that works and is available now.

The first Raspberry Pi was an immediate success, selling over 500,000 units in the first 6 months after its release in February 2012. Since then around 30 million Raspberry Pis have been sold. The current version, Version 4 Model B, is a significant upgrade, the most powerful model yet, featuring:

- 2 USB 2 ports
- 2 USB 3 ports
- 2 micro HDMI ports, supporting 2 4K displays
- 1 dedicated gigabit Ethernet port
- Supports RAM from 2 GB–8 GB
- Broadcom BCM2711, 1.5 GHz Quad-core Cortex-A72 (ARM v8) 64-bit SoC
- 2.4 GHz and 5.0 GHz IEEE 802.11ac WiFi
- Bluetooth
- 40-pin GPIO header

Compared to modern Intel and AMD CPUs, these specs are far from bragworthy, but it's enough horsepower to run a Linux graphical desktop, play music, movies, surf the web, write documents…it is quite capable for its size and price.

The Raspberry Pi is developed and produced by the Raspberry Pi Foundation, a registered nonprofit charity. The foundation supports numerous education projects for teachers and students; visit *https://raspberrypi.org* for current educational materials and information.

18.1 Starting and Shutting Down Raspberry Pi

Problem

You don't see a power switch on your Raspberry Pi (RPi), and you want to turn it on and off.

Solution

Power it up by plugging in the power connector. Shut it down from the menu in your operating system, then unplug it.

Discussion

When you shut down your RPi, you have to disconnect and then reconnect the power to start it up again.

It wouldn't surprise me if there is a power switch out there somewhere made for the RPi, though I have not seen one. One alternative is to plug it into a switched power strip.

See Also

- *https://raspberrypi.org*

18.2 Finding Hardware and How-Tos

Problem

You bought a Raspberry Pi 4 Model B, and you want to know what other hardware devices you need in order to use it.

Solution

Presumably you already have a computer monitor or TV, mouse, and keyboard. You also need:

- Raspberry Pi power supply
- HDMI-to-micro-HDMI cable
- Micro SD card of at least 16 GB
- Cooling fan, or heat sinks on the CPU, RAM module, and USB controller
- Case
- Micro SD card reader

Start by installing Raspberry Pi OS to your micro SD card on another computer. While that is running, assemble your hardware. When everything is ready, connect the power and watch your new system boot up. (See Recipes 18.4 and 18.5 to learn how to install Raspberry Pi OS.)

To learn all about RPi, there are numerous sources of schematics, specifications, howtos, and clever ideas. The "See Also" section of this recipe lists a nice selection to get you started.

Discussion

If your display does not have an HDMI port, see Recipe 18.6.

You may use any micro SD card at least 16 GB in size. High-speed cards make a noticeable performance improvement. 16 GB is pretty small, and you can use as large a card as you like.

The RPi 4B has three RAM options, 2 GB, 4 GB, and 8 GB. It is not upgradable, so whatever you buy is what you will have. 2 GB is more than enough for an internet gateway, and using your Pi as a lightweight Linux desktop. More memory is good for multimedia processing, compiling code, games, and other memory-intensive tasks.

The RPi 4B is quite a bit more powerful than older Pis, and it runs hotter than older versions. See Recipe 18.3 to learn how to keep it cool.

With 4 USB ports, the RPi 4B has a lot of flexibility. You can use standard USB keyboards and mice, a keyboard with a trackpad, and USB-to-PS/2 adapters for older keyboards and mice. You can attach a USB hard drive for more storage or backups, and any other USB devices just like on a big computer. The 40-pin GPIO header supports attaching nearly any expansion board.

There are many nice kits that bundle everything you need to get started. My favorite store is Adafruit.com, and you can find many more listed on *https://raspberrypi.org*.

See Also

These sites publish excellent Raspberry Pi tutorials:

- The Raspberry Pi Foundation
- Adafruit
- MagPi
- Hackspace
- Maker Pro
- Makezine

18.3 Cooling the Raspberry Pi

Problem

Your Raspberry Pi feels hot to the touch, and you want to install some kind of cooling for it.

Solution

Install a cooling fan in the case, or heat sinks on the CPU, RAM module, and USB controller. Install a fan and heat sinks on the Raspberry Pi 4, which runs hotter than older Pis.

Use the built-in *vcgencmd* command to measure CPU temperatures before and after installing cooling devices, and before and after running computationally intensive tasks such as compiling code or playing videos. The following example is a fanless board at idle with the case lid removed, and then after five minutes of playing a movie at 1080p:

```
$ vcgencmd measure_temp
temp=48.3'C

$ vcgencmd measure_temp
temp=61.9'C
```

This example shows the effect of a cooling fan when playing the same movie:

```
$ vcgencmd measure_temp
temp=52.1'C
```

The temperature should not go over 70°C. 40°C to 60°C is a good operating range.

See Also

- The Raspberry Pi Foundation
- Adafruit
- MagPi

18.4 Installing Raspberry Pi OS with Imager and dd

Problem

You have your hardware, and you want to install the operating system.

Solution

You will create a bootable micro SD card on another computer, then load the SD card into your Raspberry Pi and start it up.

There are four ways to get a bootable SD card:

- Use the Raspberry Pi Imager (currently only for *.deb* systems such as Debian, Ubuntu, or Mint)
- Use the NOOBS installer (Recipe 18.5)
- Copy your installation image to your micro SD card with the *dd* command
- Buy a micro SD card with the installer already loaded

I favor NOOBs because it works on all Linuxes, and it creates a rescue boot mode.

To install the Raspberry Pi Imager from the *.deb* package, download it from *https://raspberrypi.org*. Then install the package:

```
$ sudo dpkg -i imager_1.5_amd64.deb
```

Alternatively, Ubuntu users can install the Raspberry Pi Imager with *apt*:

```
$ sudo apt install rpi-imager
```

Plug your SD card into your computer. Locate it with *lsblk -p* (see Recipe 10.9).

Start the Raspberry Pi Imager from your system menu, and enjoy its cheery raspberry logo. Click Operating System to select the operating system you want to install, and the Imager will download it and copy it to your SD card. If you have already downloaded an image, scroll down the selection menu to Use Custom to select your downloaded image. You will see a screen like Figure 18-1.

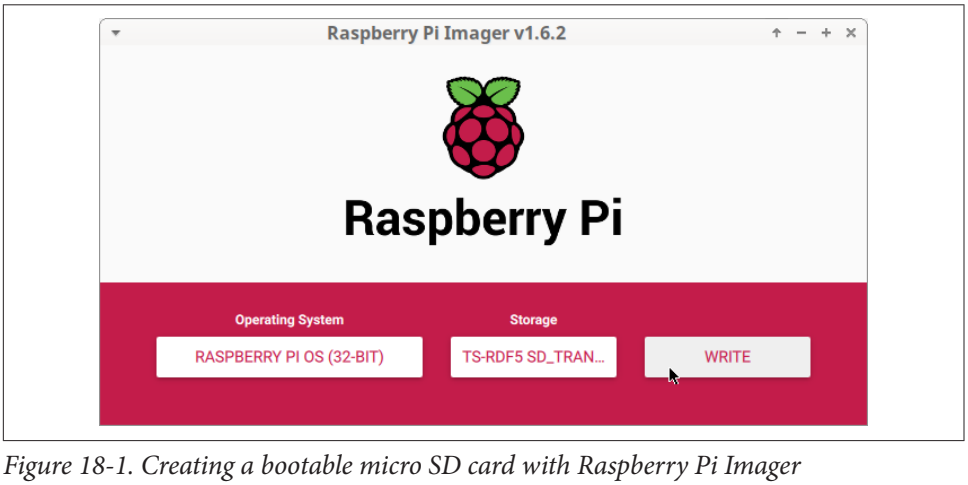


Figure 18-1. Creating a bootable micro SD card with Raspberry Pi Imager

Click SD Card to select the device, and then click Write to install the operating system.

If you don't have an Ubuntu system, download your chosen operating system image from *https://raspberrypi.org* and use the *dd* command to copy it to your SD card. The following example unzips and copies *2021-03-24-raspios-buster-armhf.zip* to the SD card:

```
$ sudo unzip -p 2021-03-24-raspios-buster-armhf.zip | \
  sudo dd of=/dev/foo bs=4M conv=fsync status=progress
```

When the SD card is ready, insert it in your Raspberry Pi and power it on. When it boots up, you will complete a short setup, and then it is ready to use.

The default user is *pi*. Look in */home/pi/Bookshelf* for a PDF copy of *The Official Raspberry Pi Beginner's Guide* by Gareth Halfacree (Raspberry Pi Press).

Discussion

You do not have to format your SD card before copying with *dd*.

The imager creates two partitions: a 256 MB FAT32 */boot* partition, and an Ext4 *rootfs* partition just big enough to hold the filesystem. On my test system that is 3.4 GB, and the rest of the card is unallocated space.

When you boot up your new system for the first time, the root filesystem is expanded to fill all the unallocated space. You may shrink the root filesystem and create more partitions with GParted (Chapter 9) or *parted* (Chapter 8).

See Also

- The Raspberry Pi Foundation

18.5 Installing Raspberry Pi with NOOBS

Problem

You want to use NOOBS to install Raspberry Pi.

Solution

NOOBS (New Out Of the Box Software) is an older installer. It works on all Linux distributions and creates a recovery boot mode, which Raspberry Pi Imager does not do.

Download NOOBS on another computer, unzip the download archive, copy all of its files to a micro SD card, then load the SD card into your Raspberry Pi and start it up.

Download NOOBS from *https://raspberrypi.org*. There are two versions: NOOBS and NOOBS Lite. NOOBS includes Raspberry Pi OS, and a network installer for other operating systems. NOOBS Lite includes only the network installer.

Unpack NOOBS after downloading:

```
$ unzip NOOBS_lite_v3_5.zip
```

Plug your SD card into your computer. Locate it with *lsblk -p* (see Recipe 10.9).

Format your micro SD card as a single FAT32 partition.

Copy all the NOOBS files to your SD card, then use it to boot your Raspberry Pi. First you will see a cool rainbow colored screen, then NOOBS boots to the installation menu (Figure 18-2). Set up networking to use the network installer or to get updates after installation. Select your operating system, then go find something to do until it is finished.

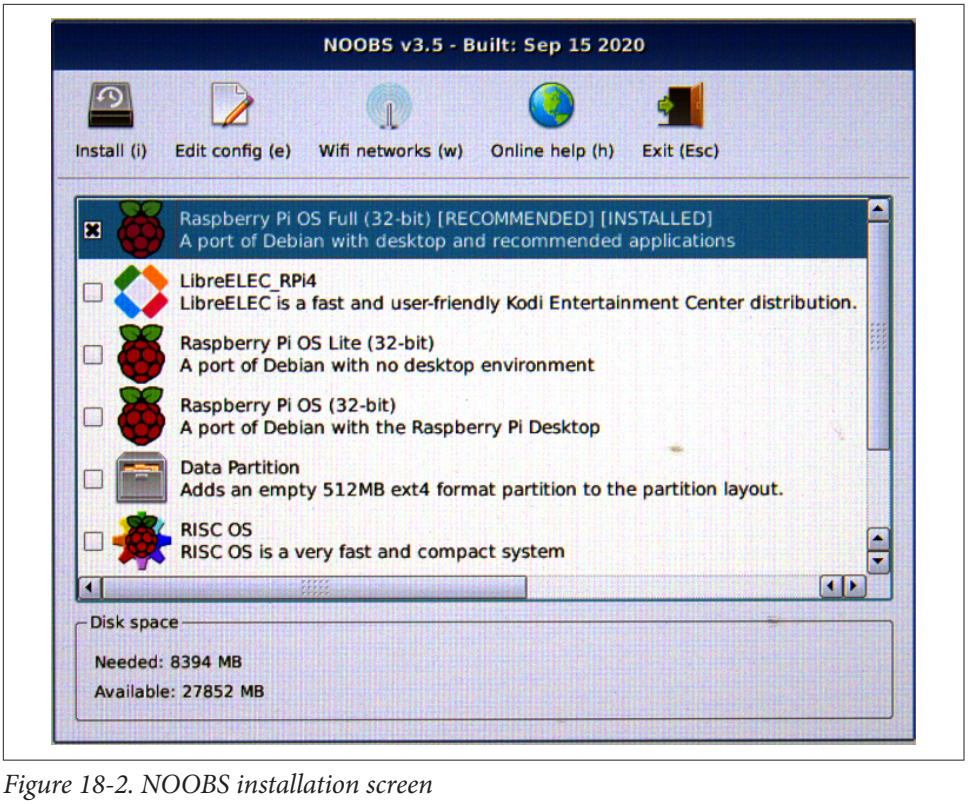


Figure 18-2. NOOBS installation screen

After installation is complete, you will go through a brief setup process, and then your Raspberry Pi is ready to use.

The default user is *pi*. Look in */home/pi/Bookshelf* for a PDF copy of *The Official Raspberry Pi Beginner's Guide* by Gareth Halfacree (Raspberry Pi Press).

Discussion

NOOBS is a simple unzip and copy, so it works on any computer.

Copying the NOOBS files to your SD card can take a long time. Usually it is faster to copy the ZIP file to the SD card, then unzip it on the card. You can delete the ZIP file after installation.

See Also

- Chapter 9
- The Raspberry Pi Foundation

18.6 Connecting to a Video Display Without HDMI

Problem

You have a TV or computer monitor that does not have HDMI ports, and you want to connect your Raspberry Pi to it.

Solution

There are four ways to connect a screen to the RPi 4B. You can use:

- Small screens made for the Raspberry Pi
- DVI-to-HDMI adapter
- VGA-to-HDMI adapter
- RCA composite video (use cables made for Raspberry Pi)

There is quite a variety of screens made for the RPi: touchscreens, LED, LCD, OLED, eInk—you name it, there is probably an RPi version. Follow the installation instructions that come with the screen.

The DVI-to-HDMI and VGA-to-HDMI adapters connect to your screen, and then your HDMI-to-micro-HDMI cable connects to the adapter. When you connect to a single HDMI display on the RPi 4B, plug into the HDMI 0 port, which is the one closest to the power port.

Composite video requires some extra steps on the RPi 4B because it is disabled by default. The easy way is to find an HDMI screen to use when you boot your Pi for the first time and complete your installation. After your installation is complete, open the configuration tool to enable composite video:

```
$ sudo raspi-config
```

Arrow-key down to 6 Advanced Options, then select A8 HDMI/Composite. Select V2 Enable Composite. Exit *raspi-config*, shut down your Pi, connect your Pi to your composite video screen, then start it up again. It should default to your composite video screen.

Discussion

Older flat-panel screens usually have VGA and composite video connectors, so you could use either composite video or a VGA-to-HDMI adapter. Technically, HDMI delivers a higher-quality image than composite video, but most people cannot tell the difference.

Figure 18-3 shows a Rocketfish DVI-to-HDMI adapter, a composite cable bundle, a Rasberry Pi 4B in a CanaKit case with the top removed, and a micro SD card.

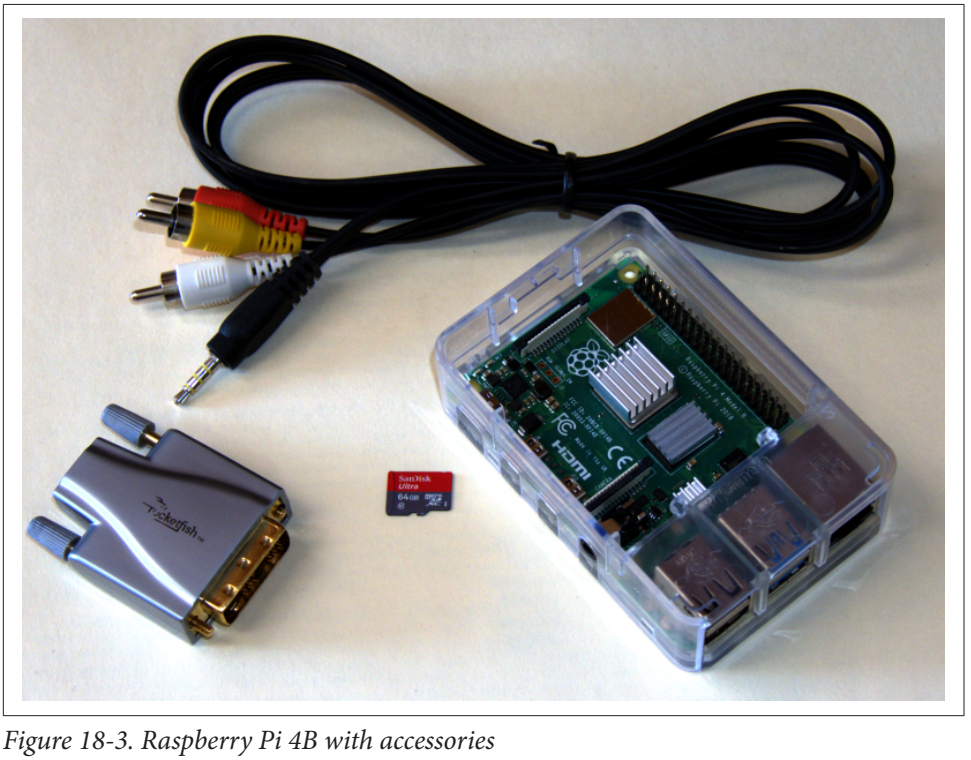


Figure 18-3. Raspberry Pi 4B with accessories

The RCA composite audio/video cable set, the one with the yellow, red, and white connectors, plugs into the nifty little 3.5 mm TRRS port on the RPi. You need one made for the RPi because there is no standard for how the TRRS plug is ordered. You might find some that don't work as they are supposed to, which is yellow for video, and red and white for audio. Avoid composite cables made for camcorders and MP3 players, which have their own weird ordering that puts the ground ring in the wrong place. The TRRS (tip-ring-ring-sleeve) plug should be configured like this:

```
Tip          Ring 1        Ring 2   Sleeve
Left audio   Right audio   Ground   Video
```

There are several options for tweaking your composite video settings in */boot/config.txt*. If you plan to use composite video, use the NOOBS installer. Then if you need to correct its settings, you can boot to recovery mode (Recipe 18.7) and have access to */boot/config.txt*.

sdtv_mode= sets the TV standard. *sdtv_mode=0* is the default for North America. Most of the world uses PAL; see Table 18-1 for the settings.

Table 18-1. sdtv_mode settings

value	mode
0	Normal NTSC (default)
1	Japanese version of NTSC
2	Normal PAL
3	Brazilian version of PAL
16	Progressive scan NTSC
18	Progressive scan PAL

sdtv_aspect= command defines the aspect ratio (Table 18-2).

Table 18-2. sdtv_aspect screen ratio settings

value	ratio
1	4:3 (default)
2	14:9
3	16:9

See Also

- Video options in config.txt

18.7 Booting into Recovery Mode

Problem

You want to know how to boot into recovery mode in case you ever have problems.

Solution

You must have installed your operating system with NOOBS because that is the only installer that sets up a recovery mode. Power on your Pi and watch your startup screen. A screen with the Raspberry Pi logo and a "For recovery mode, hold Shift" message appears briefly. Press and hold the shift key until the recovery screen appears (Figure 18-2; the recovery screen is the same as the NOOBS installation screen).

The recovery screen is a nice graphical utility for performing some basic operations. You can connect to the internet, browse the online help, edit the */boot/cinfig.txt*, or blow away your installation and install something else.

Discussion

The recovery screen is the same as the NOOBS installation screen. You have this recovery option only when you install Raspberry Pi with NOOBS, though by the time you read this it could be different.

See Also

- The Raspberry Pi Foundation

18.8 Adding a Second Ethernet Interface

Problem

You want to use your Raspberry Pi as an internet firewall/router, but it has only one Ethernet port, and you really want two Ethernet ports.

Solution

There are two ways to get a second Ethernet port: with a USB-to-Ethernet adapter or by installing an Ethernet port that connects to the GPIO pins.

USB-to-Ethernet is easy; just plug it in.

A wired Ethernet port is a little more work. You need an Ethernet adapter powered by an ENC28J60 Ethernet controller module (Figure 18-4).

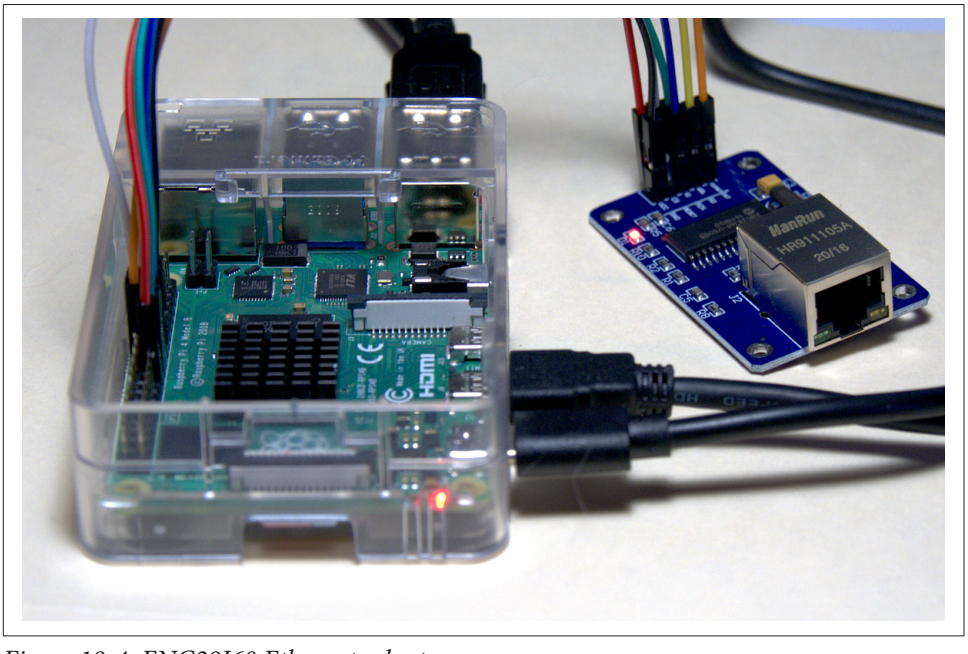


Figure 18-4. ENC28J60 Ethernet adapter

The HanRun HR911105A adapter in the photo needs seven female-to-female jumper wires to connect to the GPIO pins. You can buy multicolor bundles that supply an assortment of wires for cheap.

Before connecting the wires, generate your own handy pinout diagram by running the *pinout* command on your Raspberry Pi (Figure 18-5).

Figure 18-6 shows the diagram from RaspberryPi.org numbers and labels the GPIO pinouts.

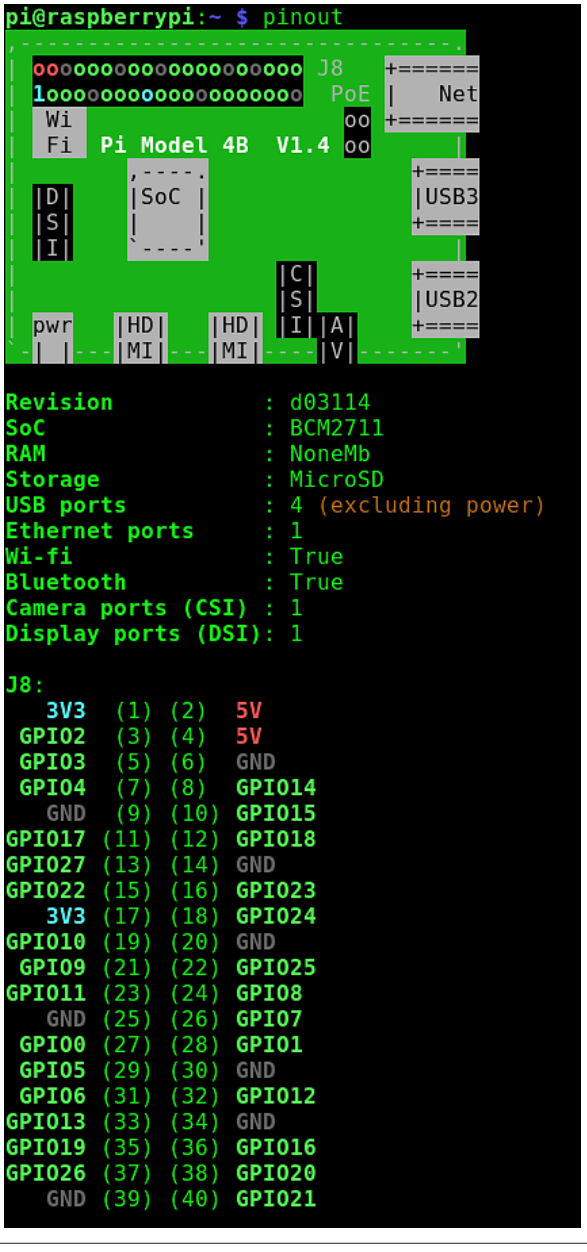


Figure 18-5. Pinout diagram generated by pinout command

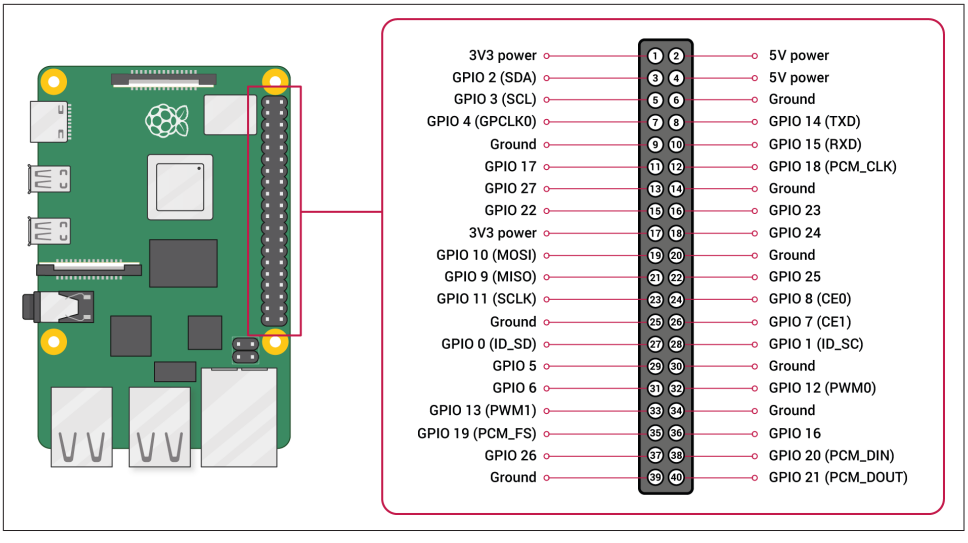


Figure 18-6. Pinout diagram from RaspberryPi.org

Edit */boot/config.txt* to enable the new Ethernet port and load the drivers:

```
dtparam=spi=on
dtoverlay=enc28j60
```

Keep a copy of the pinout diagram available and power off your Raspberry Pi. Connect the jumper wires in the following locations on your RPi and ENC28J60 module:

```
RPi           ENC28J60
---------------------------
+3V3          VCC
GPIO10        SI
GPIO9         SO
GPIO11        SCK
GND           GND

GPIO25        INT
CE0#/GPIO8    CS
```

Note the orientation of the GPIO pins: pin #1 is on the same end of the RPi board as the SD card slot. Start with 3V3 pin 17, then all of your wires are in the same area, and the other 3V3 pin is free for a case fan.

When all of the wires are in place, power up your RPi. Run *ifconfig* to see your new Ethernet interface. It should be *eth1*:

```
$ ip link show dev eth1
2: eth1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel
state DOWN mode DEFAULT group default qlen 1000
    link/ether d0:50:99:82:e7:2b brd ff:ff:ff:ff:ff:ff
```

There it is, all ready to be configured.

Discussion

The ENC28J60 Ethernet controller supports only 10 MBps. If your internet speeds are no better 10 MBps, then this is fine. On the Raspberry Pi 4 you will get as much as 900 MBps with a USB 3.0 Ethernet adapter.

Older Raspberry Pis have slower Ethernet because it runs over the USB 2.0 bus. The RPi 4B is the first Pi to have a dedicated Ethernet bus.

Keep an eye on new product releases for dual gigabit Ethernet options, as there are likely to be some soon.

The *pinout* command is provided by the *python3-gpiozero* package. This is installed by default in the Raspberry Pi OS desktop image, but not on Raspberry Pi OS Lite. Install it with *apt install python3-gpiozero*.

See Also

- GPIO diagram
- Technical information and datasheet for ENC28J60 controller

18.9 Setting Up an Internet Connection Sharing Firewall with firewalld

Problem

You want to configure a simple firewall on your Raspberry Pi that shares your internet connection and keeps the bad bits out.

Solution

We will use *firewalld* to filter incoming packets, allowing only responses to traffic that originated from inside the LAN and disallowing external connection requests.

Your internet gateway setup looks something like Figure 18-7.

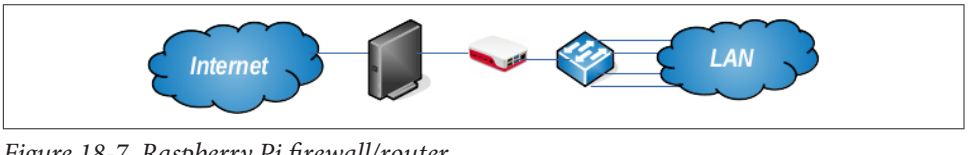


Figure 18-7. Raspberry Pi firewall/router

The big bad internet comes in through whatever device connects you to your ISP, which connects to your Raspberry Pi, which filters and routes traffic to your LAN through an Ethernet switch.

You need two network interfaces on your RPi, one that connects from your internet box to the RPi, and a second interface to connect your RPi to your LAN. In this recipe we will use two Ethernet interfaces.

Install *firewalld*, and optionally *firewall-config* and *firewall-applet*. *firewall-config* provides a graphical configuration tool, and *firewall-applet* sits in the panel and provides quick access to some commands, such as the panic button and lockdown:

```
$ sudo apt install firewalld firewall-config firewall-applet
```

Find your default router/gateway:

```
$ ip r show
default via 192.168.1.1 dev eth0 proto dhcp src 192.168.1.43 metric 303 mtu 1500
192.168.1.0/24 dev eth0 proto dhcp scope link src 192.168.1.43 metric 303 mtu
1500cat
```

default via 192.168.1.1 is your default gateway.

Make *eth1* the external interface by connecting it to your internet box. *eth0* is your internal interface, connected to your LAN switch. The two interfaces should be on different subnets. *eth1* should be on the same subnet as your internet box. Suppose the LAN interface of your internet box is 192.168.1.1, then *eth1* could be 192.168.1.2.

eth0 is on your LAN subnet, for example, 192.168.2.1.

Configure the two interfaces in */etc/dhcpcd.conf*:

```
# external interface
interface eth1
static ip_address=192.168.1.2/24
static routers=192.168.1.1

# internal interface
interface eth0
static ip_address=192.168.2.1/24
static routers=192.168.1.1
```

Reboot to apply the changes.

The next step is to set up *firewalld*. The two interfaces must be in two different firewall zones. *eth1* goes in the *external* zone, and *eth0* goes in the *internal* zone, then verify your changes:

```
$ sudo firewall-cmd --zone=external --change-interface=eth1
success
pi@raspberrypi:~ $ sudo firewall-cmd --zone=internal --change-interface=eth0
success
pi@raspberrypi:~ $ sudo firewall-cmd --get-active-zones
```

```
external
  interfaces: eth1
internal
  interfaces: eth0
```

List the configuration for each zone:

```
$ sudo firewall-cmd --zone=external --list-all
external (active)
  target: default
  icmp-block-inversion: no
  interfaces: eth1
  sources:
  services: ssh
  ports:
  protocols:
  masquerade: yes
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:

$ sudo firewall-cmd --zone=internal --list-all
internal (active)
  target: default
  icmp-block-inversion: no
  interfaces: eth0 wlan0
  sources:
  services: dhcpv6-client mdns samba-client ssh
  ports:
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

Note that *ssh* access is the default for the external zone. You may add or remove any services that you wish. You must leave *masquerade* enabled because this is what enables internet access.

Make your changes permanent:

```
$ sudo firewall-cmd --runtime-to-permanent
success
```

IPv4 forwarding is also enabled by default in the *external* zone, which you can verify by reading */proc*. IPv4 forwarding enables routing; otherwise, all the packets entering your RPi would not be routed to other hosts on your network.

```
$ cat /proc/sys/net/ipv4/ip_forward
1
```

1 means it is enabled, 0 is not enabled.

Discussion

IPv4 masquerading is network address translation, or NAT. NAT was created to extend the limited pool of IPv4 addresses (which is officially exhausted). NAT allows us to freely use the private IPv4 address spaces on our internal networks without having to purchase public IPv4 addresses. Your internet provider gives you at least one public IPv4 address. Your private addresses are translated to appear as your one public IPv4 address; otherwise, your internal hosts would have no internet access.

You can add and remove services from your zones; see Chapter 14.

See Also

- Chapter 14
- Debian Bug report logs - #914694

18.10 Running Your Raspberry Pi Headless

Problem

You are using your Raspberry Pi as an internet firewall/router, LAN router, or some kind of lightweight LAN server, and you want to reduce the load by running it without a graphical desktop.

Solution

Follow these steps:

1. Set up SSH access on your RPi (Chapter 12).
2. Run *raspi-config* to disable the graphical desktop.
3. Reboot.
4. Launch *sudo raspi-config*, then navigate to 1 System Options → S5 Boot / Auto Login (see Figure 18-8).
5. Set the next boot to console, not GUI, then reboot.

As long as you can open an SSH session to your RPi, you will be able to access it without a screen.

You can launch your graphical environment from the text console by typing the **`startx`** command.

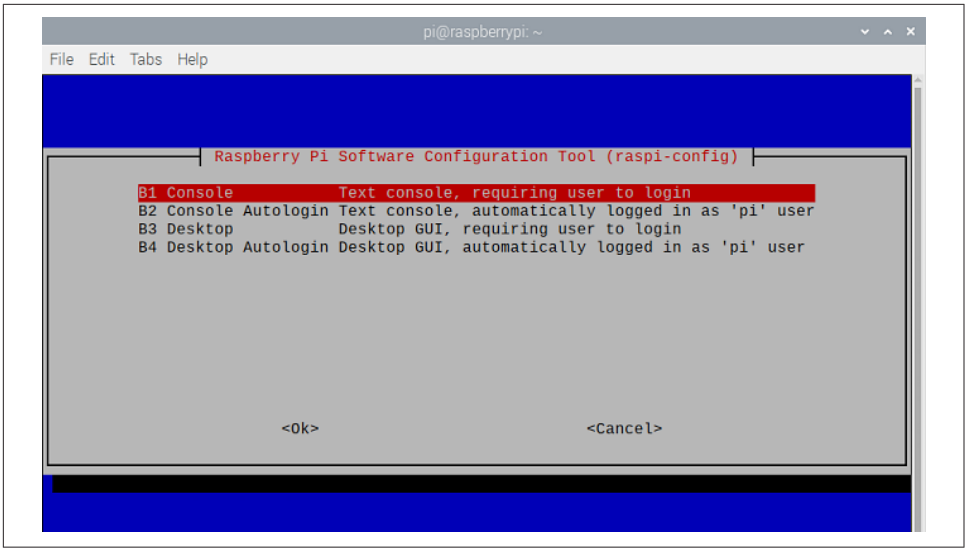


Figure 18-8. Set the next boot to console, not GUI

Discussion

rpi-config uses the *ncurses* interface. *ncurses* is a console interface that looks like a simple GUI.

When you run the Raspberry Pi headless, it needs only power and network connections, because you control it over an SSH session from another computer.

See Also

- Chapter 12
- The Raspberry Pi Foundation

18.11 Building a DNS/DHCP Server with Raspberry Pi

Problem

Your internet box does not offer much in the way of administration features, and you want control of your local name services, DNS and DHCP.

Solution

Disable the name services on your internet box, and set up a second Raspberry Pi to provide your LAN name services with Dnsmasq (Chapter 16). Use DHCP to supply

all services and addresses to your LAN hosts, including static addresses, except for your internet gateway, which should be independent of any internal services.

Discussion

You could install your name server on your internet firewall/gateway, but it is not a good security practice to put internal services on a host that is directly connected to the internet. Your Raspberry Pi DNS/DHCP server needs only a single network interface, like any other LAN server.

See Also

- Chapter 16

CHAPTER 19

System Rescue and Recovery with SystemRescue

A SystemRescue DVD or USB drive is an essential tool, and you may use it to rescue nonbooting Linux and Windows systems. In this chapter you will learn how to create SystemRescue boot media, how to find your way around SystemRescue, customize boot options, repair GRUB, retrieve files from a failing disk, reset Linux and Windows passwords, and convert SystemRescue from a read-only filesystem to a read-write filesystem with a data partition.

Any live Linux can serve as a rescue Linux. The advantage of SystemRescue is its small size, and it is tailored for rescue operations.

The root SystemRescue filesystem is read-only, so any changes you make are lost when you shut it down. You will learn how to set it up to preserve your changes, such as configurations, appearance, and adding software.

SystemRescue was originally based on Gentoo Linux, and since version 6.0 it is built from Arch Linux. Arch Linux is known for being a reliable and efficient Linux distribution, and has first-rate documentation. Visit *https://archlinux.org* for documentation and forums.

I prefer USB devices for SystemRescue, both thumb drives and USB hard disks. They're fast and have as much capacity as you need for copying files.

19.1 Creating Your SystemRescue Bootable Device

Problem

You want to make a SystemRescue DVD or USB drive.

Solution

Download the latest SystemRescue *.iso* from *https://system-rescue.org*.

The most reliable way to create a bootable SystemRescue USB stick is with the *dd* command (see Recipe 1.6). See Recipes 1.4 and 1.5 for instructions on creating a bootable DVD. Chapter 1 also describes how to boot to your new medium and how to disable Secure Boot. SystemRescue does not include signing keys, so you must disable Secure Boot.

When you boot from a USB stick, plug it in to a USB port on your computer, not a USB hub, because it will not be recognized on a hub.

Discussion

Remember to reenable Secure Boot when you are finished using SystemRescue.

See Also

- *https://system-rescue.org*
- Chapter 1

19.2 Getting Started with SystemRescue

Problem

You have booted up SystemRescue, and it stops at a plain console prompt, and you need to know what to do.

Solution

SystemRescue gives you instructions on the initial login screen (Figure 19-1). You are automatically logged in as root, and there is no root password.

You can either work from the console or type **`startx`** to launch the Xfce4 desktop environment (Figure 19-2).

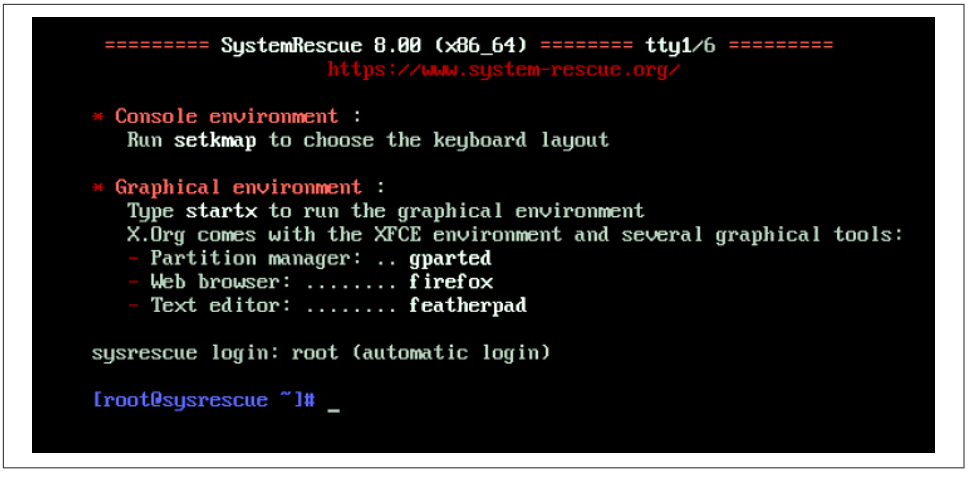


Figure 19-1. The SystemRescue login screen

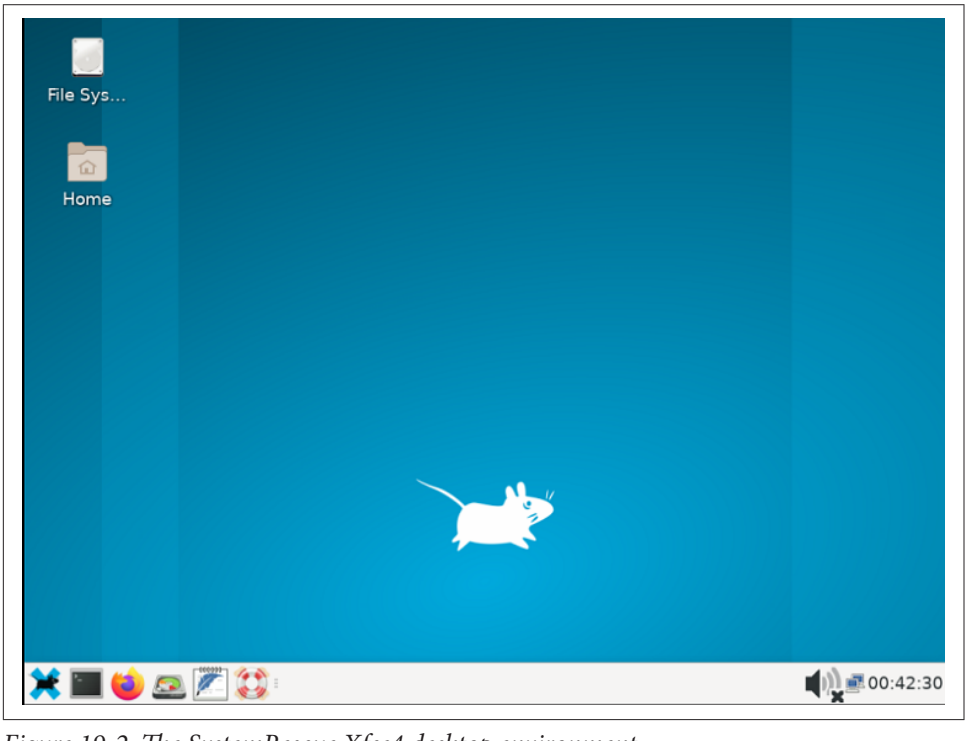


Figure 19-2. The SystemRescue Xfce4 desktop environment

Explore the applications menu, adjust the appearance of Xfce, connect to networks with NetworkManager, and shut it down or reboot just like any Linux system.

Discussion

The one thing you cannot do with the stock SystemRescue image is make persistent changes in its own configuration. Any changes you make will not survive a restart because SystemRescue runs from a compressed read-only SquashFS filesystem. However, you can set it up to preserve your changes; see Recipe 19.14.

SquashFS is the basis of many live Linux distributions, such as Ubuntu, Debian, Mint, Fedora, and Arch. It is also used by the open source router firmware projects DD-WRT and OpenWRT. SquashFS is lightweight and fast.

I like working from Xfce4 because it provides a lightweight graphical environment and applications, and an X terminal for command-line operations.

See Also

- *https://system-rescue.org*
- *https://xfce.org*

19.3 Understanding SystemRescue's Two Boot Screens

Problem

While testing SystemRescue, you noticed there are two different boot screens, and you want to know what they are for.

Solution

There are two different boot screens according to how you boot SystemRescue. You'll see one boot screen with legacy BIOS boot (Figure 19-3) and a different screen with UEFI boot (Figure 19-4).

When the UEFI setup on my Dell system is configured to allow booting legacy devices, the one-time boot menu (press F12 at startup) shows all possible boot options (Figure 19-5). SystemRescue supports UEFI, so it is not necessary to enable legacy boot. (Remember that Secure Boot must be disabled for SystemRescue to boot.)

Your system's BIOS/UEFI may not look like mine, as they all vary.

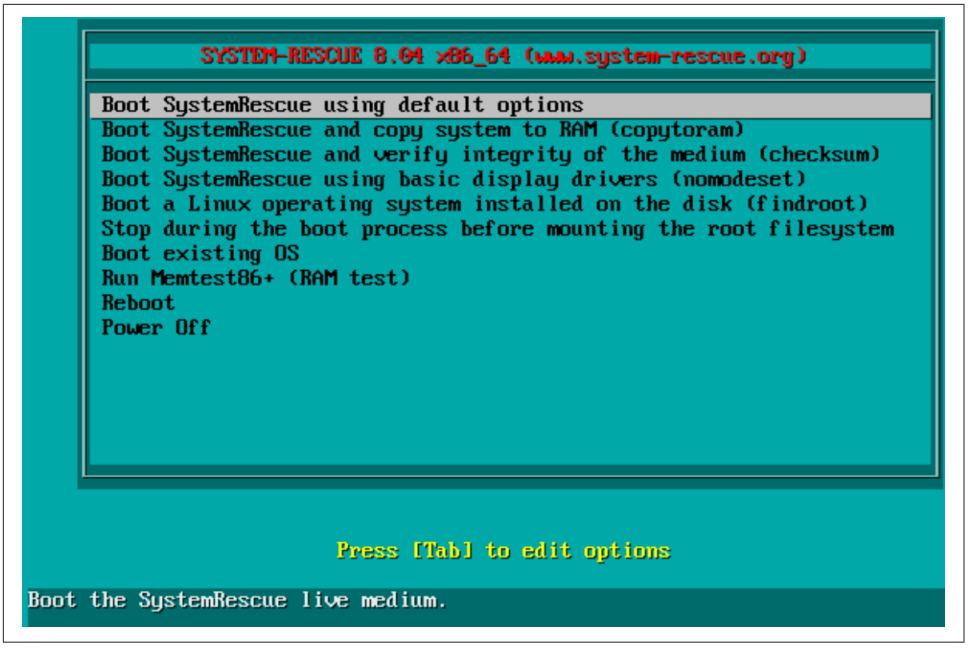


Figure 19-3. The SystemRescue legacy BIOS boot screen

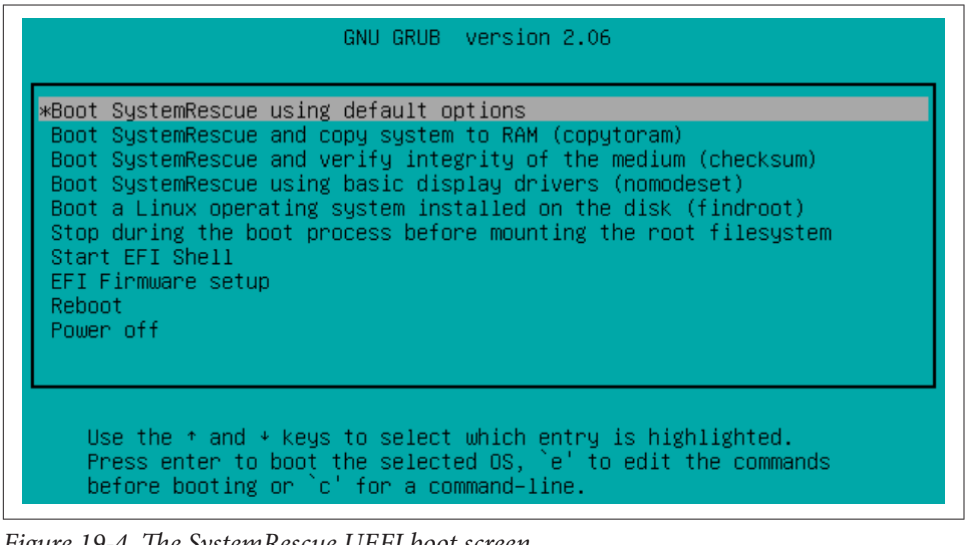


Figure 19-4. The SystemRescue UEFI boot screen

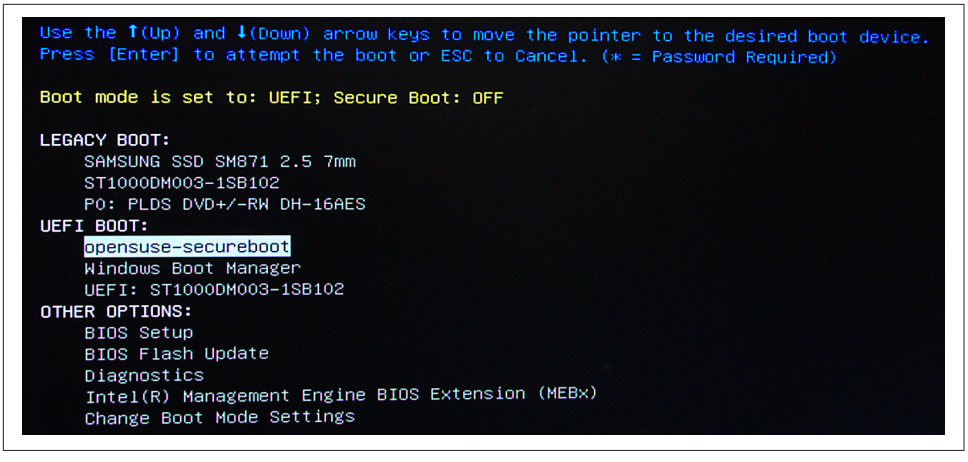


Figure 19-5. Dell one-time boot menu shows all possible boot options

The main difference between the two boot screens is the SystemRescue UEFI boot screen has the "Start EFI Shell" and "EFI Firmware Setup" options, which are not applicable to a legacy BIOS boot.

The legacy BIOS boot screen does not have the two EFI options, but it includes Memtest86+ for testing system memory.

See Recipe 19.4 to learn what the different SystemRescue boot options are for.

Discussion

You can configure removable media as the default boot device. Then you don't have to hassle with entering your BIOS/UEFI to enable a different boot device or waiting for the right moment to call up your one-time boot menu; just insert your removable boot media when you need it.

If you are running an older system with no UEFI, you won't have to choose or hassle with Secure Boot.

See Also

- *https://system-rescue.org*
- Chapter 1
- Recipe 19.4

19.4 Understanding SystemRescue's Boot Options

Problem

You want to know what all those SystemRescue boot options are for (Recipe 19.3).

Solution

The boot menu options are shortcuts for the most commonly used boot options, saving you the trouble of opening the editing form and typing them. In most cases you will use the first boot option, "Boot SystemRescue Using Default Options."

The second option, "Boot SystemRescue and copy to RAM (copytoram)," speeds up performance by loading SystemRescue completely into memory. This is especially useful when you run SystemRescue from a DVD. It uses about 2 GB of RAM.

The third option, "Boot SystemRescue and verify integrity of the medium (checksum)," tests itself for corruption. Use this to verify that your SystemRescue is healthy.

The fourth option, "Boot SystemRescue using basic display drivers (nomodeset)," uses lower-resolution basic video drivers. Use this if your video does not look right because SystemRescue does not have the right graphics drivers.

The fifth option, "Boot a Linux operating system installed on the disk (findroot)," is a good way to test if the bootloader is the problem on a nonbooting Linux installation. It will find a bootable partition, and if there is more than one it lists all of them, and you select which one you want to use.

The sixth option, "Stop during the root process before mounting the root filesystem," is a repair mode in case SystemRescue will not boot. I think it's easier to keep some extra SystemRescue drives on hand than to try to repair a broken SystemRescue.

The UEFI boot screen has two additional options: "Start EFI Shell," and "EFI Firmware Setup." "Start EFI Shell" gives you access to the numerous EFI utilities, and "EFI Firmware Setup" takes you to your system's UEFI setup.

Then Reboot and Power off.

The BIOS boot screen has two additional options: "Boot existing OS" and "Run Memtest86+." "Boot existing OS" helps diagnose bootloader problems by bypassing the system's bootloader, and "Memtest86+" tests system memory.

Both screens include "Reboot" and "Power off," for rebooting or shutting down SystemRescue instead of starting it up.

Discussion

I don't see much value in using the EFI shell because it supports advanced operations far beyond what you need for rescuing a nonbooting system. See Intel's Basic Instructions for Using the Extensible Firmware Interface to learn all about it.

All of these menu options are shortcuts for some of SystemRescue's boot options, which are documented on *https://system-rescue.org*. You may append boot options to any of the SystemRescue boot menu items, which you will see in several of the recipes in this chapter.

You can do these tasks after you start SystemRescue, or pass in boot options. In the legacy BIOS screen, select your boot entry, then press the Tab key, which opens an edit field. Enter **`rootpass=`*`yourpassword`*** **`nofirewall`**, then press Enter to resume startup.

In the UEFI boot screen, press the E key to pass in your own boot options.

See Also

- *https://system-rescue.org*

19.5 Identifying Filesystems

Problem

You need to know how to identify the filesystems on your hard disks, so you are certain which ones to use in your rescue operations.

Solution

Use the good old *lsblk* command:

```
# lsblk -f
NAME   FSTYPE   FSVER LABEL     UUID                SAVAIL FSUSE% MOUNTPOINT
loop0  squashfs 4.0                                      0   100% /run/archiso/sf
s/airootfs
sda
├─sda1
└─sda2 ntfs                    5E363
sdb
├─sdb1 vfat     FAT16 BOOT      5E2F-1E75
├─sdb2 btrfs          root      02bfdc9a-b8bb-45ac-95a8
├─sdb3 xfs            home      cc8acf0b-529e-473c-b484
└─sdb4 swap     1               7a5519ae-efe6-45e6-b147
sdc    iso9660        RESCUE800 2021-03-06-08-53-50-00
```

```
└─sdc1 iso9660        RESCUE800 2021-03-06-08-53-50-00    0   100% /run/archiso/
bootmnt
```

Discussion

Using filesystem labels makes finding the correct filesystems a lot easier. You may also use labels in place of UUIDs, for example, in */etc/fstab*. See Recipe 9.4 and Chapter 11 for information on managing filesystem labels.

lsblk does not need root privileges and displays information about your block devices in almost any way you want to see it.

See Also

- Recipe 9.4
- Recipe 11.2
- Chapter 11

19.6 Resetting a Linux Root Password

Problem

You forgot your Linux root password and need to reset it.

Solution

Boot SystemRescue, then mount the correct root filesystem. In the following examples, the root filesystem is on */dev/sdb2*. Create a mountpoint in */mnt*, then mount your filesystem:

```
# mkdir /mnt/sdb2
# mount /dev/sdb2 /mnt/sdb2
```

Change from the SystemRescue root filesystem to your mounted filesystem:

```
# chroot /mnt/sdb2/ /bin/bash
:/ #
```

Reset the root password:

```
:/ # passwd root
New password:
Retype new password:
passwd: password updated successfully
:/ #
```

Type **`exit`** to return to the SystemRescue root filesystem.

Reboot, log in, and try your new password.

Discussion

You cannot recover a forgotten password, but only create a new one.

This works for any user's password.

By changing to the host system's root filesystem you can run some commands, but not all of them because it is not a complete filesystem. It is missing all the pseudo-filesystems that exist only in memory, such as *sysfs* and *proc*. See Recipe 19.9 to learn how to set up a more complete *chroot* environment.

Once upon a time you could reset a root password by deleting the password hash in */etc/shadow*. That was then, and now the *pam* subsystem is more complex and controls authorization. If you're curious, study the SystemRescue *pam* configuration to see how it is set up to allow an empty root password.

See Also

- *https://system-rescue.org*
- Chapter 5
- *man 7 pam*

19.7 Enabling SSH in SystemRescue

Problem

You want SSH access to SystemRescue.

Solution

SSH is enabled by default, and so is the firewall. Disable the firewall to allow incoming SSH sessions.

After launching SystemRescue, disable the firewall with *systemctl*:

```
[root@systemrescue ~]# systemctl stop iptables.service
```

By default, root has no password on SystemRescue. You must create one to enable an SSH session:

```
[root@systemrescue ~]# passwd root
New password:
Retype new password:
passwd: password updated successfully
```

Now you can log in to SystemRescue from another computer:

```
$ ssh root@192.168.10.101
ssh root@192.168.1.91
The authenticity of host '192.168.1.91 (192.168.1.91)' can't be established.
ECDSA key fingerprint is SHA256:LlUCEngz5NHg98xv.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.1.91' (ECDSA) to the list of known hosts.
root@192.168.1.91's password:
[root@sysrescue ~]#
```

Discussion

Every time you boot SystemRescue, it is like a brand-new system with a different SSH host key. If you reboot SystemRescue, then open a second SSH system from the same computer to SystemRescue, you will see this warning:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
```

This goes on for a few more lines, and it tells you the remedy:

```
Offending ECDSA key in /home/duchess/.ssh/known_hosts:12
  remove with:
  ssh-keygen -f "/home/duchess/.ssh/known_hosts" -R "192.168.10.101"
```

Do what is says, then you can SSH in to SystemRescue.

You may disable the firewall from the boot menu. Press the Tab key (legacy boot) or the E key (UEFI boot) to add the *nofirewall* boot option (see the boot parameters line at the bottom of the screen, Figure 19-6).

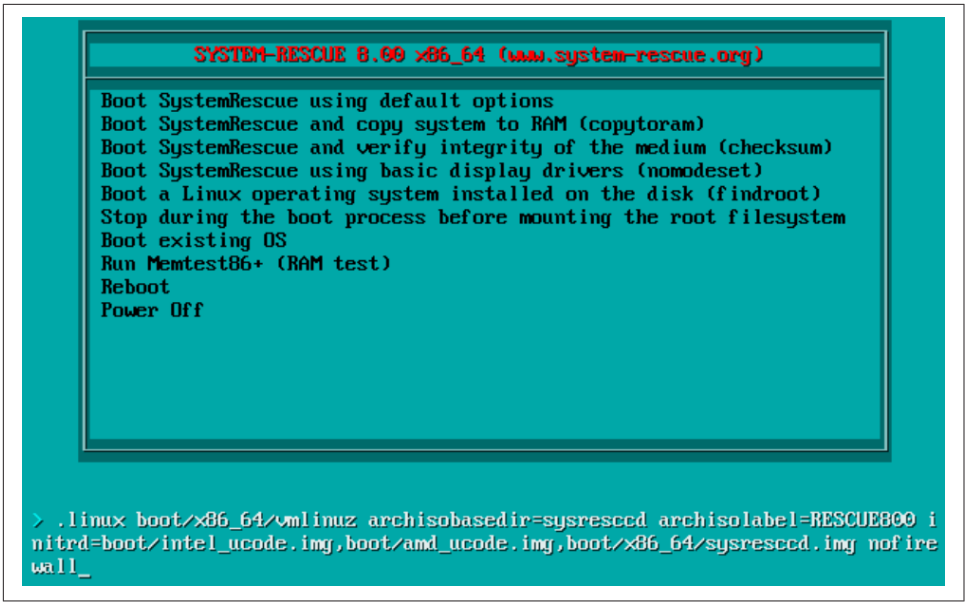


Figure 19-6. Disabling the firewall at boot

See Also

- *https://system-rescue.org*
- Recipe 19.4
- Chapter 12

19.8 Copying Files over the Network with scp and sshfs

Problem

You have SystemRescue up and running on the system you are rescuing and want to rescue files by copying them over the network.

Solution

No problem, do this just like on any Linux. First, enable SSH (Recipe 19.7). Then use *scp* or *sshfs* to move the files you want to save. All of the commands in this recipe are run from SystemRescue.

Find the filesystem you want to copy files from with *lsblk*. If you don't remember which partition you want, mount each one to see the files until you find the right one:

```
# lsblk -f
NAME   FSTYPE   FSVER LABEL     UUID                 SAVAIL FSUSE% MOUNTPOINT
loop0  squashfs 4.0                                       0   100% /run/archiso/sf
s/airootfs
sda
├─sda1
└─sda2 ntfs                    5E363E30363E0993
sdb
├─sdb1 vfat     FAT16 BOOT     5E2F-1E75
├─sdb2 btrfs          root     02bfdc9a-b8bb-45ac-95a8
├─sdb3 xfs            home     cc8acf0b-529e-473c-b484
└─sdb4 swap     1              7a5519ae-efe6-45e6-b147
sdc    iso9660        RESCUE800 2021-03-06-08-53-50-00
└─sdc1 iso9660        RESCUE800 2021-03-06-08-53-50-00    0   100% /run/archiso/b
ootmnt
sr0
```

The following example mounts the partition containing */home* on the system being rescued in */mnt* on SystemRescue, lists the files in the mountpoint, then copies the whole */home* directory to Duchess's PC with *scp*:

```
# mkdir /mnt/sdb3
# mount /dev/sdb3 /mnt/sda3
# ls /mnt/sdb3
bin   dev  home  lib64       media  opt   root  sbin  sys  usr
boot  etc  lib   lost+found  mnt    proc  run   srv   tmp  var
# scp -r /mnt/sdb3/home/ duchess@pc:
```

The result is */home/duchess/home*.



Always Create a Subdirectory in /mnt

Never mount any filesystems in */mnt*; it will freeze SystemRescue. Always create subdirectories for your mountpoints.

You may copy a space-delimited list of files and mix files and directories. This example copies them to the *rescue* directory on *duchess@pc*. The remote directory must already exist:

```
# cd /mnt/sdb3/home/
# scp -r file1.txt directory1 file2.txt duchess@pc:rescue/
```

sshfs is convenient because it mounts a remote filesystem so that it appears as a local filesystem, and you can copy files just as though they were local. Create a mountpoint on SystemRescue, then mount a remote directory on it, which must already exist. You will copy files from SystemRescue to the remote system:

```
# mkdir /mnt/remote
# sshfs duchess@pc:rescue/ /mnt/remote/
```

```
# ls /mnt/remote
rescue
```

Now you can use the *cp* command on SystemRescue, or use a graphical file manager to copy files (Figure 19-7).

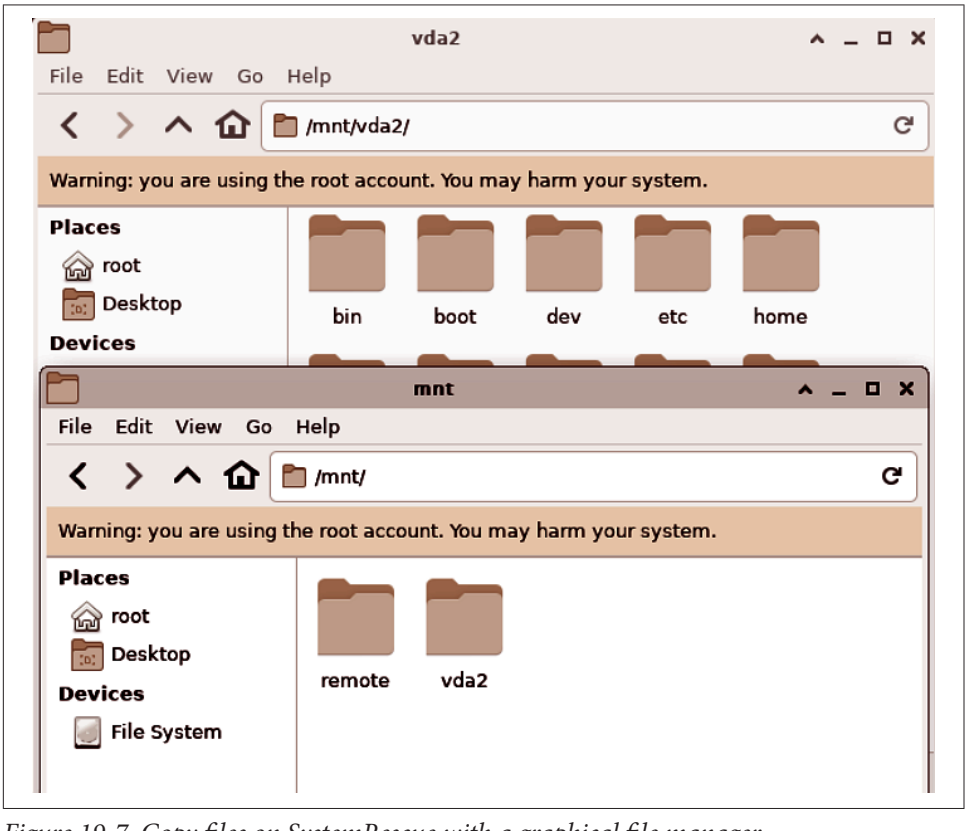


Figure 19-7. Copy files on SystemRescue with a graphical file manager

When you are finished, run **`fusermount -u remote`** to cleanly unmount the *sshfs* filesystem.

Discussion

If you have disabled SSH password authentication on the system you are copying the files to, reenable it temporarily by commenting out *PermitRootLogin no* in */etc/ssh/sshd_config*.

The syntax for connecting to the remote directory is relative to the user account you are logging into. *duchess@pc:* is the same as *duchess@pc:/home/duchess*. *duchess@pc:/*

accesses the root filesystem. When you need to edit system configuration files, use *duchess@pc:/etc*; for bootfiles, use *duchess@pc:/boot*, and so on.

You can create the remote directory from SystemRescue with *ssh*:

```
# ssh duchess@pc
duchess@pc's password:
duchess@pc:~$ mkdir remote
```

See Also

- Recipe 6.5
- Chapter 12

19.9 Repairing GRUB from SystemRescue

Problem

Your GRUB bootloader is broken, and your system does not boot.

Solution

Boot up SystemRescue, create a *chroot* environment, and reinstall GRUB.

After starting up SystemRescue, create a *chroot* environment for the root filesystem on the host:

```
# mkdir /mnt/linux
# mount /dev/sda2 /mnt/linux
# mount -o bind /proc /mnt/linux/proc
# mount -o bind /dev /mnt/linux/dev
# mount -o bind /sys /mnt/linux/dev
```

Enter the *chroot* environment:

```
# chroot /mnt/linux /bin/bash
:/ #
```

If */boot* is on its own partition, mount it:

```
:/ # mount /dev/sda1 /boot/
```

Now reinstall GRUB:

```
:/ # grub-install /dev/sda
```

When this is completed, type **`exit`** to leave the *chroot* environment, then unmount all the filesystems. Reboot your system, and GRUB should work.

Discussion

Be very careful when creating a *chroot* environment, and make sure you are using the correct partitions and filesystems. *chroot*, change root, is a great utility for changing to a different root filesystem without rebooting.

You must unmount all *chroot* filesystems so that they unmount cleanly. It should be all right to simply reboot, but it is cheap insurance to manually unmount them first.

See Also

- *man 1 chroot*

19.10 Resetting a Windows Password

Problem

You lost your Windows password, and you don't want to jump through the usual Windows hoops to reset it.

Solution

No worries, for SystemRescue will have you back in action in a jiffy. Boot up SystemRescue on your Windows machine, then mount your Windows system directory:

```
# mkdir /mnt/windows
# mount /dev/sda2
```

Navigate to the */mnt/windows/Windows/System32/config* directory, then use the *chntpw* (change NT password) command to list users:

```
# cd /mnt/windows/Windows/System32/config
# chntpw -l SAM
chntpw version 1.00 140201, (c) Petter N Hagen
Hive <SAM> name (from header): <\SystemRoot\System32\Config\SAM>
ROOT KEY at offset: 0x001020 * Subkey indexing type is: 686c <lh>
File size 65536 [10000] bytes, containing 7 pages (+ 1 headerpage)
Used for data: 318/31864 blocks/bytes, unused: 29/12968 blocks/bytes.

| RID -|---------- Username ------------| Admin? |- Lock? --|
| 01f4 | Administrator                  | ADMIN  |          |
| 03e9 | duchess                        | ADMIN  |          |
| 01f7 | DefaultAccount                 |        | dis/lock |
| 01f5 | Guest                          |        | dis/lock |
| 01f8 | WDAGUtilityAccount             |        | dis/lock |
```

Examine the information on the user you want to change:

```
# chntpw -u Administrator SAM
chntpw version 1.00 140201, (c) Petter N Hagen
Hive <SAM> name (from header): <\SystemRoot\System32\Config\SAM>
ROOT KEY at offset: 0x001020 * Subkey indexing type is: 686c <lh>
File size 65536 [10000] bytes, containing 9 pages (+ 1 headerpage)
Used for data: 321/33816 blocks/bytes, unused: 34/27336 blocks/bytes.

================ USER EDIT ====================

RID     : 0500 [01f4]
Username: Administrator
fullname:
comment : Built-in account for administering the computer/domain
homedir :

00000220 = Administrators (which has 2 members)

Account bits: 0x0210 =
[ ] Disabled        | [ ] Homedir req.    | [ ] Passwd not req. |
[ ] Temp. duplicate | [X] Normal account  | [ ] NMS account     |
[ ] Domain trust ac | [ ] Wks trust act.  | [ ] Srv trust act   |
[X] Pwd don't expir | [ ] Auto lockout    | [ ] (unknown 0x08)  |
[ ] (unknown 0x10)  | [ ] (unknown 0x20)  | [ ] (unknown 0x40)  |

Failed login count: 0, while max tries is: 0
Total  login count: 5

- - - - User Edit Menu:
 1 - Clear (blank) user password
 2 - Unlock and enable user account [probably locked now]
 3 - Promote user (make user an administrator)
 4 - Add user to a group
 5 - Remove user from a group
 q - Quit editing user, back to user select
Select: [q] ^
```

Enter **1** to remove the existing password:

```
Select: [q] ^ 1
Password cleared!
[...]
```

Press **q** to quit, and **y** to "write hive files," which saves your changes.

Now the Administrator, or whatever user you selected, must log in and set a new password.

Discussion

You cannot create a new password or recover the old one with *chntpw*, but only delete it. Then you log in without a password and create a new password, or, if your user is present, let them do it.

See Also

- *https://system-rescue.org*
- *man 8 chntpw*

19.11 Rescuing a Failing Hard Disk with GNU ddrescue

Problem

You suspect that your hard disk is dying, and you want to copy your data from it before it expires.

Solution

You want the excellent GNU *ddrescue* utility. *ddrescue* tries to copy all the good blocks first, saving as much data as possible, and skips over the bad blocks, tracking their locations in a log file. You may make multiple passes to try to capture more data.

The disk you are trying to rescue must be unmounted. You need another disk, also unmounted, such as a USB storage device or an internal hard disk, to copy the rescued data to. Your target partition must already exist and be at least 50% larger than the partition you are trying to rescue.

The following example copies */dev/sdb1* to */dev/sdc1*:

```
# ddrescue -f -n /dev/sdb1 /dev/sdc1 ddlogfile
GNU ddrescue 1.25
Press Ctrl-C to interrupt
     ipos:   100177 MB, non-trimmed: 0 B    current rate:   207 MB/s
     opos:   100177 MB, non-scraped: 0 B    average rate: 83686 kB/s
non-tried:    47868 MB,  bad-sector: 0 B,     error rate:      0 B/s
  rescued:   100177 MB,   bad areas: 0,         run time:    23m 56s
pct rescued:    66.77%, read errors: 0,   remaining time:      6m 4s
                        time since last successful read:         0s
Copying non-tried blocks... Pass 1 (forwards)
```

This will take some time. When it is finished the last line will say "Finished."

This example does one pass, copying the most easily read blocks as quickly as possible. This is a good tactic on a drive with a lot of errors because *ddrescue* won't spend a

lot of time trying to recover the most damaged blocks. After making this first pass, run it three more times to try to recover more data:

```
# ddrescue -d -f -r3 /dev/sdb1 /dev/sdc1 ddlogfile
```

When it is finished, run a filesystem check on the recovery disk, which should remain unmounted. This example checks and automatically repairs Ext4 filesystems:

```
# e2fsck -vfp /dev/sdc1
```

-f forces a check, in case *e2fsck* thinks the filesystem is clean. *-p* is short for preen, or repair, and *-v* is verbose. If it finds a problem that requires your intervention, it prints a description of the problem and exits.

e2fsck -vf [device] launches an interactive check and repair.

fsck.vfat -vfp [device] is for FAT16/32.

xfs_repair [device] is for XFS filesystems.

If your recovered filesystem passes the filesystem checks, go ahead and copy your files to their final location. If there are still problems, mount it read-only:

```
# mkdir /mnt/sdc1-copy
# mount -o ro /dev/sdc1 /mnt/sdc1-copy
```

Then copy as many files as you can to another disk.

Discussion

Make sure you have GNU *ddrescue*, by Antonio Diaz Diaz, and not *dd-rescue* by Kurt Garloff. *dd-rescue* is a great tool, but it is more complicated to use.

ddrescue copies at the block level, so it doesn't matter what the filesystem is that you are trying to rescue. *ddrescue* will make an exact copy regardless of what filesystems your Linux supports.

If *ddrescue* runs out of space, it will fail at the very end, so be sure your recovery drive has a generous amount of room.

You can use *ddrescue* on USB sticks, CompactFlash, and SD cards.

See Also

- GNU ddrescue
- *man 8 fsck (e2fsprogs)*

19.12 Managing Partitions and Filesystems from SystemRescue

Problem

You want to partition your hard disk or make changes to filesystem, and you need to do it from an external Linux system.

Solution

Use SystemRescue. SystemRescue includes both GParted and *parted*. You don't have to mount any filesystems, and SystemRescue will operate directly on the block devices on your host system. Use *lsblk* to see your host block devices:

```
[root@systemrescue ~]# lsblk -p -o NAME,FSTYPE,LABEL
NAME          FSTYPE     LABEL
/dev/loop/0   squashfs
/dev/sr0
/dev/sr1      iso9660    RESCUE800
/dev/sda
├─/dev/sda1   vfat
├─/dev/sda2   xfs        osuse15-2
├─/dev/sda3   xfs        home
├─/dev/sda4   xfs
└─/dev/sda5   swap
/dev/sdb
└─/dev/sdb1   xfs        backups
/dev/sr0
```

Follow the recipes in Chapters 8, 9, and 11 for managing partitions and filesystems.

See Also

- Chapter 8
- Chapter 9
- Chapter 11

19.13 Creating a Data Partition on Your SystemRescue USB Drive

Problem

SystemRescue on USB drives is working well for you, but you want to know how to partition your device with the SystemRescue root filesystem on the first partition and a writable filesystem on the second partition. Then you will need only a single device for copying files.

Solution

You can do this in just a few steps.

The stock SystemRescue image cannot boot from a partition. It uses less than a gigabyte of space, so any USB stick will have a lot of wasted space. The trick to making SystemRescue bootable from a partition is to make your SystemRescue ISO capable of booting from a partition, add a Master Boot Record (MBR), then install the boot code, *mbr.bin*.

You need *isohybrid* and *mbr.bin*, which are provided by *syslinux*. This is provided by the *syslinux* package on Fedora and openSUSE, and by *syslinux-utils* and *install-mbr* on Ubuntu.

In the following examples, replace */dev/sdc* with your own device.

First, make the SystemRescue image bootable from a partition:

```
$ isohybrid --partok systemrescuecd-8.01-amd64.iso
```

Create an *msdos* partition table on your USB drive. Use GParted (Recipe 9.2) or *parted*:

```
$ sudo parted /dev/sdc
(parted) mklabel msdos
```

Create two partitions on your USB drive. Set the filesystem type on Partition 1 as FAT32, and set the *boot* flag. The following example creates an approximately 2 GB boot partition:

```
(parted) mkpart "sysrec" fat32 1MB 2000MB
(parted) set 1 boot
```

Add a second partition for data storage, using any filesystem type you want. The following example creates a 2 GB partition, then quits *parted*:

```
(parted) mkpart "data" xfs 2001MB 4000MB
(parted) q
```

Create your filesystems. Partition 1 is FAT32, with the label *SYSRESCUE*; Partition 2 is XFS, labeled *data*:

```
$ sudo mkfs.fat -F 32 -n SYSRESCUE /dev/sdc1
$ sudo mkfs.xfs -L data /dev/sdc2
```

Install SystemRescue to the first partition:

```
$ sudo dd status=progress if=systemrescuecd-8.01-amd64.iso of=/dev/sdc1
```

On Ubuntu, install the MBR to the USB drive:

```
$ sudo install-mbr /dev/sdc
```

On other Linuxes, use *dd*:

```
$ sudo if=/usr/share/syslinux/mbr.bin of=/dev/sdc
```

mbr.bin may be in a different directory, depending which Linux you are using.

Boot up your SystemRescue drive, and it should start up normally.

Discussion

Your filesystems do not need labels; they are a convenience to help you remember what they are for.

You may use any filesystem on the first partition. FAT32 is universal, so you can boot SystemRescue on Linux, macOS, and Windows. For copying files from macOS and Windows, format your data partition as FAT32 or exFAT.

You will have to mount your second partition manually, then you can use it however you want. It is convenient for copying files from the host system, and the coolest thing about having this writable partition is you can use it as a backing store for storing changes you make to SystemRescue, such as configuration changes and installing software. See Recipe 19.14 to learn all about it.

See Also

- Chapter 8
- Chapter 11
- *https://system-rescue.org*
- *man 1 isohybrid*
- *man 1 dd*

19.14 Preserving Changes in SystemRescue

Problem

SystemRescue is working well for you, but you wish you could preserve some changes and not have to start over every time you start SystemRescue.

Solution

See Recipe 19.13 to learn how to create a writable partition on your SystemRescue USB drive. Give this partition a filesystem label, such as *data*. Once this is set up, boot up SystemRescue, select your boot menu choice, press the Tab key, and append **`cow_label=`*`data`*** to your boot selection (Figure 19-8).

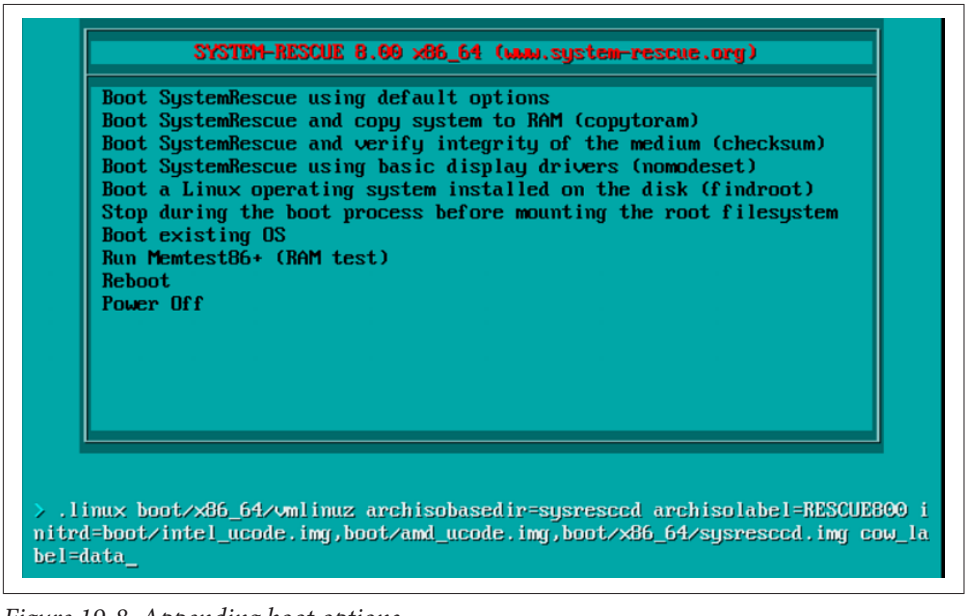


Figure 19-8. Appending boot options

SystemRescue mounts both of your partitions in */run/archiso/*:

```
# lsblk -p
lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
[...]
sdc      8:32   1   3.7G  0 disk
├─sdc1   8:33   1     2G  0 part /run/archiso/bootmnt
└─sdc2   8:34   1   152G  0 part /run/archiso/cowspace
```

All changes you make to SystemRescue are stored in */run/archiso/cowspace/persistent_RESCUE800*, and the root filesystem remains the same. *RESCUE800* is different for each SystemRescue release, according to the release number.

You can configure SystemRescue just like any Linux: enable and disable services, set a root password, change appearance, install new software, change network configurations, or write new documents.

Discussion

Large USB drives are inexpensive, and needing only a single rescue device is convenient. You can use either USB thumb drives or USB hard drives.

See Also

- *https://system-rescue.org*

CHAPTER 20

Troubleshooting a Linux PC

Linux includes numerous utilities to help diagnose and fix problems, enough to fill several thick books. In this chapter, the focus is on using system logs to find out what went wrong, building a central systemd logging server, monitoring hardware health, finding and stopping troublesome processes, getting the best performance from hardware, and tips and tricks for diagnosing hardware issues.

Overview

Get to know your system logs well, and you will find the causes of problems. If learning the cause does not point you to a solution, you have the information you need to ask for help, whether it's product documentation, distribution documentation, paid support, or community support.

Get well-acquainted with the documentation for your Linux distributions, especially changelogs and release notes, and the documentation for the servers and applications that you use. Ubuntu, Fedora, and openSUSE all excel at maintaining their documentation and detailed release notes. Also get acquainted with the forums, wikis, and chats for your distro, servers, and applications. For every issue you encounter, it is likely many other users have dealt with the same issue.

Prevention

Most errors are caused by software. Even consumer-grade hardware is pretty robust, and it fails most often from abuse and age. The most common hardware failures are components with moving parts:

- SATA and SCSI disk drives
- CPU coolers

- Power supplies
- Case fans
- CD/DVD drives

There are simple measures you can take to extend the life of your hardware. Overheating and unreliable electricity are electronics killers. Good cooling is essential for keeping your computers healthy. Good cooling comes from well-designed cases that provide proper airflow, heat sinks and CPU coolers, and case fans oriented so that air is pulled in and pushed out correctly. This can get a little noisy, and you can buy cases, power supplies, and fans that run quietly. Vacuum out your computer insides periodically with a nonstatic vacuum, and clean case filters. If you prefer using compressed air to blow the dust out, be careful with fans. If you spin them too fast you can damage their bearings.

A power conditioner provides continuous protection from voltage sags and spikes, and from radio and electromagnetic interference. Surge protectors cost less, but only provide protection from surges. Voltage sags are just as damaging as spikes. A power conditioner more than pays for itself in longer life and stable operation.

Patience

Patience is your best friend when debugging problems. It's best to take it slowly and systematically:

- Review instructions and make sure you did not miss any steps or make a mistake.
- Is an update available? Many times this is the solution.
- Copy error messages and log file entries, and use them in web searches and trouble tickets.
- What are the last things that happened before the error occurred? What are the exact steps that created the error? Is it reproducible?
- Are the last things that happened reversible? If the answer is yes, undo them one at a time and test. Making multiple changes at once means you may not discover what caused the error.
- As a last resort, reboot. Really! This works a surprising number of times to resolve the issue, though you may not learn what the problem was.

Some graphical applications, such as the wonderful digiKam photo manager and editor, emit all manner of details when you start them from a terminal, as this snippet shows after digiKam failed to start:

```
$ digikam
Object::connect: No such signal org::freedesktop::UPower::DeviceAdded(QString)
Object::connect: No such signal org::freedesktop::UPower::DeviceRemoved(QString)
```

```
digikam: symbol lookup error: digikam: undefined symbol:
_ZNK11KExiv2Iface14AltLangStrEdit8textEditEv
```

I have no idea what this means, but someone does, so I can reference this in a web search, or ask for help in the digiKam forums.

When you ask for help, remember patience and courtesy. When you are asked for additional information, provide exactly what you are asked for. When you resolve your issue, share your solution and thank the people who helped you.

20.1 Finding Useful Information in Logfiles

Problem

Weird stuff is happening, and you need to know where to start figuring it out.

Solution

Log everything, and then read your logfiles. */var/log* contains log files, and the *dmesg* and *journalctl* commands display log messages. systemd manages all logs with *journald*, so you will see a lot of duplication with *dmesg* and */var/log*.

dmesg reads the kernel ring buffer, which is a special memory location for recording kernel activity. Look in *dmesg* to see everything that happened at startup; hardware activity after startup, such as attaching and removing USB devices; and network interface activity. The kernel ring buffer is a fixed size, so new entries overwrite the oldest entries. Nothing is lost because the kernel logs are stored in */var/log/messages*, */var/log/dmesg*, and *journalctl*.

Read *dmesg* like this:

```
$ dmesg | less
[    0.000000] microcode: microcode updated early to revision 0x28,
date = 2019-11-12
[    0.000000] Linux version 5.8.0-45-generic (buildd@lcy01-amd64-024) (gcc
(Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0, GNU ld (GNU Binutils for Ubuntu) 2.34)
#51~20.04.1-Ubuntu SMP Tue Feb 23 13:46:31 UTC 2021
(Ubuntu 5.8.0-45.51~20.04.1-generic 5.8.18)
[...]
```

When you are looking for something specific, use *grep*, like when you are having trouble with a storage drive:

```
$ dmesg | grep -w sd
[11236.888910] sd 7:0:0:0: [sdd] Attached SCSI removable disk
[11245.095341] FAT-fs (sdd1): Volume was not properly unmounted. Some data may
be corrupt. Please run fsck.
```



Finding Whole Words with grep

When you want to find a word with *grep* use the *-w* switch. For example, when you grep for *ping* you will get results like piping, escaping, sleeping. *-w* returns only whole word matches.

Run *dmesg -T* to see human-readable timestamps:

```
$ dmesg -T | less
[Tue Mar 23 15:25:17 2021] PCI: CLS 64 bytes, default 64
[Tue Mar 23 15:25:17 2021] Trying to unpack rootfs image as initramfs...
[Tue Mar 23 15:25:17 2021] Freeing initrd memory: 56008K
[...]
```

The default is the seconds and nanoseconds since startup. Run *dmesg --follow* to monitor new events as they occur, such as plugging and unplugging USB devices. Press Ctrl-C to stop.

Look for certain logging levels, such as errors and warnings:

```
$ dmesg -l err,warn
```

Run *dmesg -h* to see commands and options.

/var/log is the legacy location for log files, and you will still find logs there, depending on how your Linux distribution manages them. */var/log* is easy to search because most of the files are in plain text. When you're not sure where to start looking, *grep* the whole directory.

For example, suppose you thought you installed *graphicsmagick*, but can't find it. Take a quick look in */var/log*, and there it is, a record of its installation:

```
$ sudo grep -ir graphicsmagick /var/log
apt/history.log:Install: libgraphicsmagick-q16-3:amd64 (1.4+really1.3.35-1,
automatic), graphicsmagick:amd64 (1.4+really1.3.35-1)
[...]
/var/log/dpkg.log:2021-03-11 17:00:57 install libgraphicsmagick-q16-3:amd64
1.4+really1.3.35-1
[...]
```

Systemd stuffs all logging into *journalctl*, so you can use it exclusively and not bother with *dmesg* and */var/log*:

```
$ journalctl
```

Invoke it with *sudo* to see if that adds any more information. Usually it doesn't.

journalctl defaults to displaying oldest entries first. Press the spacebar or PageUp/Down keys to navigate a screen at a time, or use the arrow keys to scroll one line at a time. Ctrl-End goes all the way to the newest, and Ctrl-Home goes back to the oldest. Press the Q key to exit.

See newest entries first:

```
$ journalctl -r
```

It does not wrap long lines by default, so you have to use arrow keys to read long lines. Make it wrap by piping its output to *less*:

```
$ journalctl -r | less
```

See the most recent entries, with explanation messages, if there are any. This shows an example of an explanation message:

```
$ journalctl -ex | less

-- The unit grub-initrd-fallback.service has successfully entered the 'dead'
state.
Mar 27 10:14:29 client4 systemd[1]: Finished GRUB failed boot detection.
-- Subject: A start job for unit grub-initrd-fallback.service has finished
successfully
-- Defined-By: systemd
```

Search for specific services, such as MariaDB:

```
$ sudo journalctl -u mariadb.service
Mar 19 16:07:27 client4 /etc/mysql/debian-start[7927]: Looking for 'mysql' as:
/usr/bin/mysql
Mar 19 16:07:27 client4 /etc/mysql/debian-start[7927]: Looking for 'mysqlcheck'
as: /usr/bin/mysqlcheck
[...]
```

Select date ranges, which you can define in several ways:

```
$ journalctl -u mariadb.service -S today
$ journalctl -u ssh.service -S '1 week ago'
$ journalctl -u libvirtd.service -S '2021-03-05'
$ journalctl -u httpd.service -S '2021-03-05' -u '2021-03-09'
$ journalctl -u nginx.service -S '2 hours ago'
```

When you do not specify times, the default is 00:00:00, midnight. Specify times in HH:MM:SS format:

```
$ journalctl -u httpd.service -S '2021-03-05 13:15:00' -U now
```

Look at activity from an hour ago until 5 minutes ago and show the unit filenames:

```
$ journalctl -S '1h ago' -U '5 min ago' -o with-unit
```

journalctl sorts logs by system boot. Look at HTTP server activity since the most recent boot, and limit the number of lines displayed to the most recent 50:

```
$ journalctl -b -n 50 -u httpd.service
```

See what happened three boots ago:

```
$ journalctl -b -2 -u httpd.service
```

List all recorded boot sessions, with timestamps:

```
$ journalctl --list-boots
```

You can filter for specific severity levels. When you specify a single level, in this example, *crit*, it shows all messages from *crit* up to the most severe level, *emerg*:

```
$ journalctl -b -1 -p "crit" -u nginx.service
```

Define your own range, for example from *crit* to *warning*:

```
$ journalctl -b -3 -p "crit.."warning"
```

Follow new events as they are logged, starting from the 10 most recent entries:

```
$ journalctl -n 10 -u mariadb.service -f
```

Ctrl-C stops it.

And, of course, use good old *grep* to find things, like usernames, or any search term you wish:

```
$ journalctl -b -1 | grep madmax
```

Discussion

Severity levels are the standard syslog levels, from 0 to 7, with 0 being the most severe, and 7 the least severe:

```
emerg      (0)
alert      (1)
crit       (2)
err        (3)
warning    (4)
notice     (5)
info       (6)
debug      (7)
```

journalctl provides dozens of ways to filter and parse your output, and you can learn even more in *man 1 journalctl*.

See Also

- *man 3 syslog*
- *man 1 journalctl*
- *man 1 dmesg*
- *https://systemd.io*

20.2 Configuring journald

Problem

You're not sure what the default configuration is for *journald*, and you need to know how to see the current configuration and how to change it.

Solution

journald is configured in */etc/systemd/journald.conf*. Some default options are commented out, and all compile-time defaults are documented in *man 5 journald.conf*. We will take a look at the most commonly used options.

Storage=auto means different things on different distros. Ubuntu and Fedora use */run/log/journal/* for volatile storage, and persistent storage is in */var/log/journal*. See the locations of log files, used space, and free space with *systemctl*:

```
$ systemctl status systemd-journald.services
● systemd-journald.service - Journal Service
   Loaded: loaded (/usr/lib/systemd/system/systemd-journald.service; static;
   vendor preset: disabled)
   Active: active (running)
   [...]
Mar 27 15:04:40 server2 systemd-journald[508]: Runtime journal (/run/log/journal/
1181e27c52294e97a8ca5c5af5c92e20) is 8.0M, max 2.3G, 2.3G free.
Mar 27 15:04:55 server2 systemd-journald[508]: Time spent on flushing to /var is
381.408ms for 1176 entries.
Mar 27 15:04:55 server2 systemd-journald[508]: System journal (/var/log/journal/
1181e27c52294e97a8ca5c5af5c92e20) is 16.0M, max 4.0G, 3.9G free.
```

openSUSE puts volatile storage in */run/log/journal/* and persistent storage in */var/log/messages*. If you prefer to use */var/log/journal*, create it and change the group owner to *systemd-journal*:

```
$ sudo mkdir /var/log/journal
$ sudo chgrp /var/log/journal/ systemd-journal
```

You don't have to change anything else, and the storage is changed after reboot. Other options are *volatile*, *persistent*, and *none*.

volatile stores logs only in memory, in */run/log/journal/*.

persistent stores logs on disk and uses */run/log/journal/* when the disk is not available, such as early in system startup.

none disables all local logging, and you have the option to send log messages to a central logging server.

SystemMaxUse= controls the size of log storage on disk, and *RuntimeMaxUse*= controls the size of volatile storage. The default is 10% of available space in the filesystem, to a maximum of 4 GB.

SystemKeepFree= and *RuntimeKeepFree*= control how much disk space is left free for other uses. The defaults are 15% and 4 GB. You may change these by specifying numbers of bytes, or use K, M, G, T, P, and E; for example, 25 G (gigabytes).

MaxRetentionSec= controls how long files are retained. The default is 0, which disables it, and files are retained according to other settings, such as available disk space. You may configure a time value, using `year`, `month`, `week`, `day`, `h`, or `m`, for example, `6 month`.

Discussion

journald automatically handles log rotation. Active files are rotated into archived files, and archived files are deleted according to your configuration.

See Also

- *man 5 journald.conf*

20.3 Building a Logging Server with systemd

Problem

You want to set up a central logging server so that logs are preserved when systems go down, and for centralized management.

Solution

systemd provides a remote logging daemon, *journald*. Client machines send their log messages to the *journald* server. The prerequisites are as follows:

- A machine to host the log files
- Network access to the logging server for clients
- The *systemd-journal-remote* package installed on the log server and on all clients
- Your public key infrastructure (PKI) already in place (Recipe 13.5), with keys and certificates distributed to servers and clients

After installing *systemd-journal-remote*, edit */etc/systemd/journal-remote.conf* on the server. I like to store encryption keys and certificates in */etc/pki/journald/*:

```
[Remote]
Seal=false
SplitMode=host
ServerKeyFile=/etc/pki/journald/log-server.key
ServerCertificateFile=/etc/pki/journald/log-server.crt
TrustedCertificateFile=/etc/pki/journald/ca.crt
```

Set permissions for the server key and certificates:

```
$ sudo chmod -R 0755 /etc/pki/journald
$ sudo chmod 0440 /etc/pki/journald/log-server.key
```

Change the group owner of the server private key to *systemd-journal-remote*:

```
$ sudo chgrp systemd-journal-remote /etc/pki/journald/log-
server.key
```

Enable and start the *systemd-journal-remote* service, starting *systemd-journal-remote.socket* first:

```
$ sudo systemctl enable --now systemd-journal-remote.socket
$ sudo systemctl enable --now systemd-journal-remote.service
```

Check the status of both to make sure they started correctly. Open the necessary ports in the server firewall:

```
$ sudo firewall-cmd --zone=internal --add-port=19532/tcp
$ sudo firewall-cmd --zone=internal --add-port=80/tcp
$ sudo firewall-cmd --runtime-to-permanent
$ sudo firewall-cmd --reload
```

On every client, create a new user, *systemd-journal-upload*. This is the user that the *systemd-journal-upload* process uses to transfer log messages to the central server:

```
$ sudo useradd -r -d /run/systemd -M -s /usr/sbin/nologin -U \
systemd-journal-upload
```

Set permissions for the client key and certificates:

```
$ sudo chmod -R 0755 /etc/pki/journald
$ sudo chmod 0440 /etc/pki/journald/client.key
```

Edit */etc/systemd/journal-upload.conf* with the URL and TCP port to your log server, and the locations of the client key and certificates:

```
[Upload]
URL=https://logserver.example.com:19532
ServerKeyFile=/etc/pki/journald/client1.key
ServerCertificateFile=/etc/pki/journald/client1.crt
TrustedCertificateFile=/etc/pki/journald/ca.crt
```

Restart the *systemd-journal-upload.service*:

```
$ sudo systemctl restart systemd-journal-upload.service
```

If it restarts successfully, without errors, run the following steps to test that the client is sending log entries to the server. Check the log directory on the server:

```
$ sudo ls -la /var/log/journal/remote/
total 7204
drwxr-xr-x  2 systemd-journal-remote systemd-journal-remote  6 Mar 26 16:41 .
drwxr-sr-x+ 4 root                   systemd-journal        60 Mar 26 16:41 ..
rw-r-----  1 systemd-journal-remote systemd-journal-remote 8388608 Mar 26  1
10:46 'remote-CN=client1.example.com'
```

Looking good so far. Now, send the server a message from the client:

```
$ sudo logger -p syslog.debug "Hello, I am client1! Do you hear me?"
```

Run your favorite *journalctl* incantation on the server to call up the most recent entries. If you see the client message, you know you set it all up correctly:

```
Mar 27 18:30:11 client1 madmax[15228]: Hello, I am client1! Do you hear me?
```

Discussion

A central logging server preserves client logs and centralizes logging storage for easier maintenance and analysis. Every client has their own directory on the server.

Seal=false disables cryptographic signing of journal entries. To try it out, refer to the *--setup-keys* option in *man 1 journalctl*. I could not find a definitive answer if it provides a meaningful benefit, but it doesn't hurt to learn about it.

SplitMode=host stores each client log in their own file. Set it to *false* to dump everything into a single file.

ServerKeyFile=, *ServerCertificateFile=*, and *TrustedCertificateFile=* are where your encryption keys and certificates are stored.

See Also

- *man 5 journal-remote.conf*
- *man 5 journald.conf*
- *man 1 journalctl*

20.4 Monitoring Temperatures, Fans, and Voltages with lm-sensors

Problem

Your want to measure temperatures inside your computer case, fan speeds, and voltages.

Solution

Use *lm-sensors* to continually monitor CPU, hard disk, and case temperatures. This is supplied by the *sensors* package on openSUSE, *lm_sensors* on Fedora, and *lm-sensors* on Ubuntu.

After installing *lm-sensors*, run the *sensors-detect* command to calibrate *lm-sensors* to your hardware:

```
$ sudo sensors-detect
# sensors-detect version 3.6.0
# Board: ASRock H97M Pro4
# Kernel: 5.8.0-45-generic x86_64
# Processor: Intel(R) Core(TM) i7-4770K CPU @ 3.50GHz (6/60/3)

This program will help you determine which kernel modules you need
to load to use lm_sensors most effectively. It is generally safe
and recommended to accept the default answers to all questions,
unless you know what you're doing.

Some south bridges, CPUs or memory controllers contain embedded sensors.
Do you want to scan for them? This is totally safe. (YES/no):
[...]
```

Press Enter to accept all of the defaults. When it is finished you will see something like this:

```
To load everything that is needed, add this to /etc/modules:
#----cut here----
# Chip drivers
coretemp
nct6775
#----cut here----
If you have some drivers built into your kernel, the list above will
contain too many modules. Skip the appropriate ones!

Do you want to add these lines automatically to /etc/modules? (yes/NO) yes
Successful!
```

The modules will be loaded after a restart, or you can load them immediately:

```
$ sudo systemctl restart systemd-modules-load.service
```

Now run the *sensors* command and see what you get:

```
$ sensors
coretemp-isa-0000
Adapter: ISA adapter
Package id 0:  +42.0°C  (high = +86.0°C, crit = +96.0°C)
Core 0:        +34.0°C  (high = +86.0°C, crit = +96.0°C)
Core 1:        +35.0°C  (high = +86.0°C, crit = +96.0°C)
Core 2:        +32.0°C  (high = +86.0°C, crit = +96.0°C)
Core 3:        +31.0°C  (high = +86.0°C, crit = +96.0°C)

nouveau-pci-0300
Adapter: PCI adapter
GPU core:     +1.01 V  (min =  +0.70 V, max =  +1.20 V)
fan1:        2850 RPM
temp1:        +51.0°C  (high = +95.0°C, hyst =  +3.0°C)
                       (crit = +105.0°C, hyst =  +5.0°C)
                       (emerg = +135.0°C, hyst =  +5.0°C)

dell_smm-virtual-0
Adapter: Virtual device
Processor Fan: 1070 RPM
Other Fan:        0 RPM
Other Fan:      603 RPM
CPU:            +41.0°C
SODIMM:         +25.0°C
SODIMM:         +35.0°C
SODIMM:         +34.0°C
```

This shows information for CPU cores, a graphics adapter, fans, and memory modules. You see the current temperatures, and the high, critical, and emergency temperature ranges. CPUs have built-in self-preservation and shut down when they get too hot.

Use the *watch* command to see updated status every two seconds, with any differences highlighted:

```
$ watch -d sensors
```

Set a different update interval, like 10 seconds:

```
$ watch -d -n 10 sensors
Every 10.0s: sensors
[...]
```

Press Ctrl-C to stop.

Discussion

lm_sensors is not magic, it reads only devices that have temperature sensors and that also have Linux drivers. Most temperature sensors are not very precise, so don't worry about small fluctuations.

Monitoring temperatures, voltages, and fan speeds can give you early warning of trouble. It is cheaper to replace a fan than to rebuild a cooked computer. Voltage drops could indicate a failing power supply or a bad connection.

Before you modify the */etc/modules* file, check your kernel configuration to see if the modules suggested by *sensors-detect* are already loaded, or are statically compiled. Your kernel configuration file is in the */boot* directory, named *config-kernel-version*, like *config-5.8.0-45-generic*. For example, search for the *nct6775* module:

```
$ grep -i nct6775 config-5.8.0-45-generic
CONFIG_SENSORS_NCT6775=m
```

The *m* means it is a loadable kernel module. Check if it is already loaded:

```
$ lsmod | grep nct6775
```

If this returns nothing, go ahead and add it to */etc/modules*. If it were statically compiled, it would look like this in *config-**:

```
CONFIG_SENSORS_NCT6775=y
```

The *y* means it is built-in to the kernel, so do not add it to */etc/modules*.

See Also

- *man 1 watch*
- *man 1 sensors*
- *man 8 lsmod*
- *https://kernel.org*

20.5 Adding a Graphical Interface to lm-sensors

Problem

You want a configurable graphical display for *lm-sensors* that updates automatically.

Solution

You have several good choices. Graphical frontends to *lm-sensors* also support other monitors, such as *smartmontools* and *hddtemp*. Psensor provides a large display, colored graphs, and simple configuration to rename labels and show just what you want to see (Figure 20-1).

Psensor supports alarms. Enable alarms individually, like for the CPU cores and fans, by clicking on each monitor to bring up a Preferences menu (Figure 20-2).

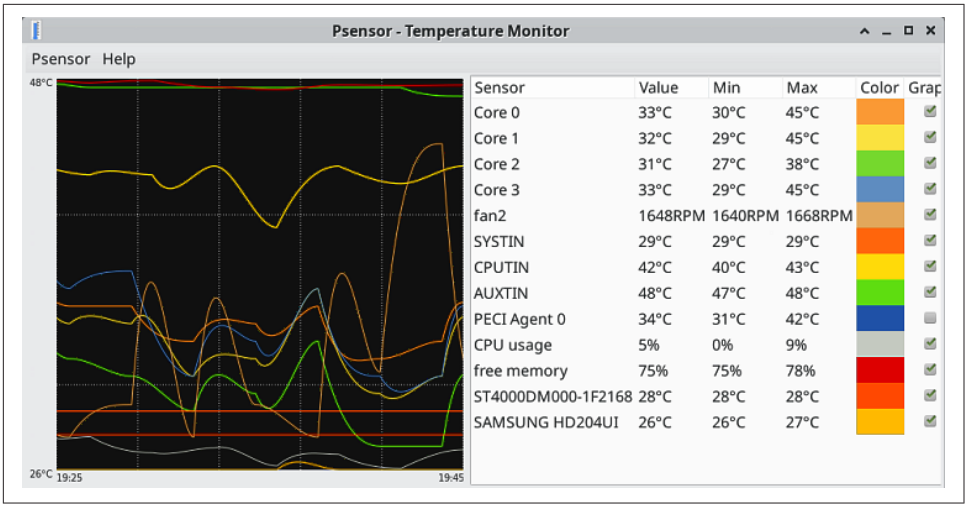


Figure 20-1. Psensor tracks multiple hardware monitors

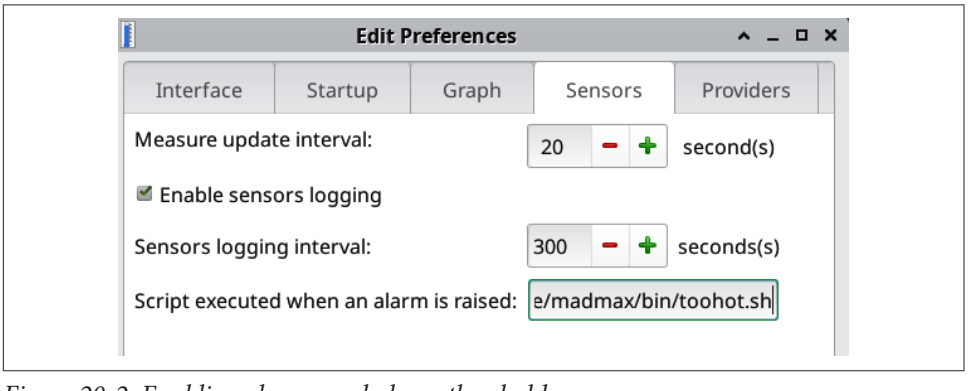


Figure 20-2. Enabling alarms and alarm thresholds

You need to write a simple script to set up an alarm, like the following example:

```
#!/bin/bash
# toohot.sh, plays a mad klavichord riff when a sensor monitor
# exceeds its upper limit

play /home/madmax/Music/klavichord-4.wav
```

Install *sox* to get the *play* command. Make your script executable:

```
$ chmod +x toohot.sh
```

Test it:

```
$ play toohot.sh
```

When it works to your satisfaction, configure Psensor to use it. Open Psensor → Preferences → Sensors (Figure 20-3).

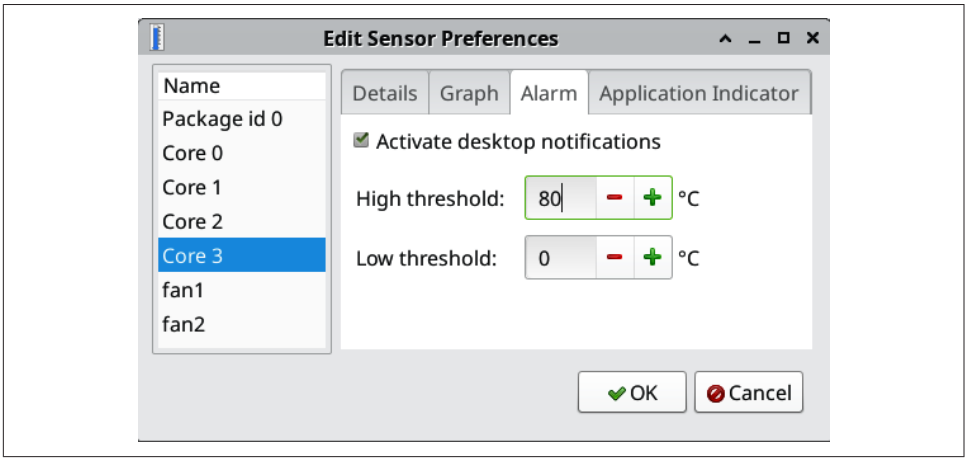


Figure 20-3. Setting up an alarm

A simple way to test it in Psensors is to set some of your maximum temperatures too low.

Many desktop environments, such as Xfce4, GNOME, and KDE, have nice little taskbar plug-ins, such as what's shown in Figure 20-4 for Xfce4.

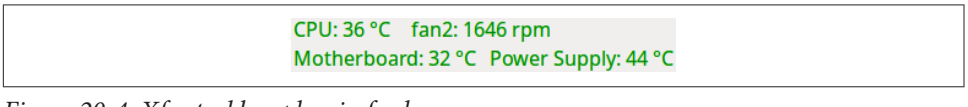


Figure 20-4. Xfce taskbar plug-in for lm-sensors

All of them have *sensor* in their package names, except *gnome-shell-extension-freon*.

Discussion

You can write your script to automatically shut the system down when an alarm is triggered, like this simple example:

```
#!/bin/bash
echo "Help, too hot, I am shutting down right now!" && shutdown -h now
```

See Also

- *man 1 play*
- *man 1 sensors*
- Psensor

20.6 Monitoring Hard Disk Health with smartmontools

Problem

You need to know when a hard disk is faulty or, preferably, when it is becoming faulty, so you can replace it before you lose data.

Solution

Most hard disks and solid-state drives come with S.M.A.R.T. (Self-Monitoring Analysis and Reporting Technology) built in. S.M.A.R.T. tracks and records certain performance attributes, which you can monitor to (hopefully) predict immiment failures. Linux users have *smartmontools* to read this information and give warnings.

smartmontools is provided by the *smartmontools* package. It should install and start a systemd service automatically, which you can check with *systemctl*:

```
$ systemctl status smartd.service
```

Use the *smartctl* command to see if your disk has S.M.A.R.T. support. Look for the *SMART* support lines:

```
$ sudo smartctl -i /dev/sda
smartctl 7.1 2019-12-30 r5022 [x86_64-linux-5.8.0-45-generic] (local build)
Copyright (C) 2002-19, Bruce Allen, Christian Franke, www.smartmontools.org

=== START OF INFORMATION SECTION ===
Model Family:     Seagate Desktop HDD.15
Device Model:     ST4000DM000-1F2168
[...]
SMART support is: Available - device has SMART capability.
SMART support is: Enabled
```

Enable and disable *smartctl* for each disk you want to monitor:

```
$ sudo smartctl -s on /dev/sda
$ sudo smartctl -s off /dev/sda
```

Use the *-x* flag for a complete data dump:

```
$ sudo smartctl -x /dev/sda
```

Run the short health check:

```
$ sudo smartctl -H /dev/sda
smartctl 7.1 2019-12-30 r5022 [x86_64-linux-5.8.0-45-generic] (local build)
Copyright (C) 2002-19, Bruce Allen, Christian Franke, www.smartmontools.org

=== START OF READ SMART DATA SECTION ===
SMART overall-health self-assessment test result: PASSED
```

Use the *-Hc* flags to see the complete report.

Check the log file:

```
$ sudo smartctl -l error /dev/sda
smartctl 7.0 2019-05-21 r4917 [x86_64-linux-5.3.18-lp152.66-preempt] (SUSE RPM)
Copyright (C) 2002-18, Bruce Allen, Christian Franke, www.smartmontools.org

=== START OF READ SMART DATA SECTION ===
SMART Error Log Version: 1
No Errors Logged
```

There is a short self-test and a long self-test. They tell you how long each test will take when you start them:

```
$ sudo smartctl -t long /dev/sda
[...]
=== START OF OFFLINE IMMEDIATE AND SELF-TEST SECTION ===
Sending command: "Execute SMART Extended self-test routine immediately in
off-line mode".
Drive command "Execute SMART Extended self-test routine immediately in off-line
mode" successful.
Testing has begun.
Please wait 109 minutes for test to complete.
Test will complete after Thu Mar 25 17:06:33 2021

Use smartctl -X to abort test.
```

It will not notify you when it is finished, and you can check the log file at any time:

```
$ sudo smartctl -l selftest /dev/sda

[sudo] password for carla:
[...]
=== START OF READ SMART DATA SECTION ===
SMART Self-test log structure revision number 1
Num  Test_Description    Status                  Remaining  LifeTime(hours)
 # 1  Extended offline    Self-test routine in progress 70%      7961
 # 2  Short offline       Completed without error       00%      7960
 # 3  Short offline       Completed without error       00%      7952
 [...]
```

Remember to update the hard drive database periodically:

```
$ sudo update-smart-drivedb
/usr/share/smartmontools/drivedb.h updated from branches/RELEASE_7_0_DRIVEDB
```

Discussion

S.M.A.R.T. is about 60% reliable. It could be better, but the S.M.A.R.T. standard leaves a lot of room for interpretation, and every drive manufacturer implements it differently. Manufacturer documentation is scarce, and the best resources I have found are Wikipedia and *https://smartmontools.org*. As always, your best insurance is regular backups.

Even so, it's free, it's easy to use, and often useful. Pay attention to the *Pre-fail* attributes (as in the following snippet), and review the introduction to this chapter on how to get better performance and reliability from your systems.

Run *sudo smartctl -a /dev/sda* to dump all the S.M.A.R.T. data. The section that tends to cause alarm is this one:

```
SMART Attributes Data Structure revision number: 10
Vendor Specific SMART Attributes with Thresholds:
ID# ATTRIBUTE_NAME          FLAG     VALUE WORST THRESH TYPE      UPDATED
  1 Raw_Read_Error_Rate     0x000f   119   099   006    Pre-fail  Always
  3 Spin_Up_Time            0x0003   092   091   000    Pre-fail  Always
  4 Start_Stop_Count        0x0032   099   099   020    Old_age   Always
  5 Reallocated_Sector_Ct   0x0033   100   100   010    Pre-fail  Always
  7 Seek_Error_Rate         0x000f   059   057   030    Pre-fail  Always
  9 Power_On_Hours          0x0032   089   089   000    Old_age   Always
 10 Spin_Retry_Count        0x0013   100   100   097    Pre-fail  Always
 12 Power_Cycle_Count       0x0032   099   099   020    Old_age   Always
183 Runtime_Bad_Block       0x0032   100   100   000    Old_age   Always
184 End-to-End_Error        0x0032   100   100   099    Old_age   Always
187 Reported_Uncorrect      0x0032   100   100   000    Old_age   Always
188 Command_Timeout         0x0032   100   099   000    Old_age   Always
189 High_Fly_Writes         0x003a   100   100   000    Old_age   Always
190 Airflow_Temperature_Cel 0x0022   072   059   045    Old_age   Always
191 G-Sense_Error_Rate      0x0032   100   100   000    Old_age   Always
192 Power-Off_Retract_Count 0x0032   100   100   000    Old_age   Always
193 Load_Cycle_Count        0x0032   096   096   000    Old_age   Always
194 Temperature_Celsius     0x0022   028   041   000    Old_age   Always
197 Current_Pending_Sector  0x0012   100   100   000    Old_age   Always
198 Offline_Uncorrectable   0x0010   100   100   000    Old_age   Offline
199 UDMA_CRC_Error_Count    0x003e   200   200   000    Old_age   Always
240 Head_Flying_Hours       0x0000   100   253   000    Old_age   Offline
241 Total_LBAs_Written      0x0000   100   253   000    Old_age   Offline
242 Total_LBAs_Read         0x0000   100   253   000    Old_age   Offline
```

The `TYPE` column tells the type of the attributes, either `Pre-fail` or `Old_age`. When you see all those `Pre-fail` and `Old_age` labels, it doesn't mean your disk is doomed, that is just the type of attribute on that row.

`Pre-fail` is a critical attribute, one that may indicate imminent failure, and it is always included in health assessments.

`Old_age` is a noncritical attribute; it is not included in disk health reports.

`ID#` and `ATTRIBUTE_NAME` identify each attribute. These vary by manufacturer.

`FLAG` is the attribute handling flag, and has no relevance to disk health.

The `VALUE` column displays the current values for the attributes. These range from 0-255, except for 0, 254, and 255. 253 means "unused," like when you have a new

drive. VALUE is a scale from good to bad, with higher numbers being good and lower numbers bad, except the temperature attributes, which are temperatures in Celsius.

WORST is the lowest value recorded for that attribute.

THRESH is the lowest threshold for each attribute, and when a Pre-fail attribute falls below THRESH, then disk failure may be imminent.

UPDATED is supposed to indicate when the attributes have been updated. Always is both online and offline, and Offline supposedly means only when offline tests are run. Usually this is inaccurate, and not all that helpful in any case.

If an attribute enters a failed state, the time it failed is recorded in *WHEN_FAILED*.

RAW_VALUE is particular to each manufacturer. Ignore it.

See Also

- *man 8 smartctl*
- *man 8 smartd*
- *man 8 update-smart-drivedb*
- *man 5 smartd.conf*

20.7 Configuring smartmontools to Send Email Reports

Problem

You want email notifications of any problems emailed to you by *smartd*.

Solution

First, check if you already have system mail set up and working by sending a test message to another user on the system, such as root:

```
$ echo "Hello, this is my message" | mail -s "Message subject" root@localhost

[root@localhost ~]# mail
"/var/mail/root": 1 message 1 unread
>U "/var/mail/root": 1 message 1 new
>N   1 stash    Mon Mar 29 15:26  13/429   Message subject
?
```

Press 1 to read the message, and q to quit. This shows that system mail is already set up. If it is not, install *mailx* and *postfix*. *mailx* is a mail user agent (MUA), which is a mail client like Evolution, Thunderbird, KMail, Mutt, and so on. *postfix* is a mail

transfer agent (MTA). You need both. After installation, check if they are running with *systemctl*:

```
$ systemctl status smartd.service
$ systemctl status postfix.service
```

If they are not, enable and start them. Then try your test message again.

```
$ sudo systemctl enable --now smartd.service
$ sudo systemctl enable --now postfix.service
```

smartd is configured in */etc/smartd.conf* or */etc/smartmontools/smartd.conf*. The default is to scan for all possible devices and email error reports to the root user. It is better to configure which devices you want monitored. Every Linux has its own special configuration. The following should work on all of them, and of course you must specify your own disks and email address:

```
DEFAULT -a -o on -S on -s (S/../../3/02|L/../../5/01):
/dev/sda
/dev/sdb
/dev/sdc
DEFAULT -H -m root -M test
```

Save your configuration changes and reload *smartd.service*:

```
$ sudo systemctl reload smartd.service
```

Discussion

The default behavior in *smartd.conf* is to scan for all available drives. It is more efficient to specify the drives you want to monitor.

The *-a* flag is the same as all of these combined:

- *-H*, check the S.M.A.R.T. health status
- *-f*, report Usage Attributes (*VALUE* and *WORST*) failures
- *-t*, report changes in Prefailure and Usage Attributes
- *-l*, report increases in ATA errors
- *-l selftest*, report increases in Self-Test Log errors
- *-l selfteststs*, report changes of Self-Test execution status
- *-C 197*, report nonzero values of the current pending sector count
- *-U 198*, report nonzero values of the offline pending sector count

That covers all the important stuff, and of course you may tweak it to report whatever attributes you like.

-M test sends the specified user (in the preceding example, *-m root@localhost*) a test message at every startup. You can remove this when you are confident it is working the way you want.

There are several packages that provide the *mail* binary: *mailutils*, *mailx*, *bsd-mailx*, and *s-nail*, to name a few. For a simple local mailer for system daemons to use, any of them will do the job, and the *mail* binary takes the same options on all of them.

You don't have to use *postfix*, but can use any MTA that you prefer, such as Exim or Sendmail.

See Also

- *man 8 smartd*
- *man 5 smartdconf*

20.8 Diagnosing a Sluggish System with top

Problem

Your system usually runs well, but now everything is bogging down and taking forever. Applications are taking a long time to start or shut down, or are slow to respond to user input. You need to find out the cause, and then fix it.

Solution

Fire up the *top* command to see which processes are using excessive system resources. CPU and memory hogs make your nice powerful system feel ancient:

```
$ top
Tasks: 284 total,   1 running, 283 sleeping,   0 stopped,   0 zombie
%Cpu(s):  6.4 us,  4.8 sy,  0.0 ni, 88.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :  15691.4 total,   6758.9 free,   4913.0 used,   4019.6 buff/cache
MiB Swap:  15258.0 total,  15258.0 free,      0.0 used.  10016.5 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S    %CPU  %MEM   TIME+ COMMAND
   1299 duchess      9   0 2803912  22296  17904 S  80.5   0.1  172:25 Web Content
   1685 duchess     20   0 3756840 543124 241296 S   7.6   3.4   27:53 firefox
  15926 libvirt+    20   0 5151504   2.3g  25024 S   1.7  15.3    1:39 qemu
  [...]
```

top runs until you stop it, refreshing every few seconds and displaying processes in order from the most to least active. Press the q key to exit.

This shows a wealth of information. The relevant bit is that `Web Content` is using 80.5% share of CPU time. Poorly constructed websites are a common cause of bringing your system to a halt. The fast way to correct this is to kill the offending process.

Process IDs are in the left column. Press the k key to open the kill dialog. If the default PID to kill is correct, press the Enter key. Press Enter to accept the default *15/sigterm* to terminate the process:

```
PID to signal/kill [default pid = 1299]
Send pid 1299 signal [15/sigterm]
```

If that does not kill the process, use the nuclear option, *9*, which is *sigkill*:

```
PID to signal/kill [default pid = 1299]
Send pid 1299 signal [15/sigterm] 9
```

If you do not have sufficient permissions to kill the process, start *top* with *sudo*. Or, run *sudo kill <pid>* in another terminal.

Discussion

Killing a process is not always the best solution. If the process is controlled by systemd, systemd may immediately restart it, or other processes may be dependent on it, and then you risk making a mess. If possible, shut it down with *systemctl stop <service name>*. If the offending process is not controlled by systemd, then go ahead and kill it.

The default PID to kill is always the one at the top, the one using the most system resources. If that is not the one you want to stop, you can enter a different PID.

What is this *sigterm* stuff, you ask? *Signals* are inherited from Unix, and are very crufty, with numerous variations added over the years that you can learn all about in *man 2 signal*, such as *SIGHUP*, *SIGINT*, *SIGQUIT*, and a host of others.

The two that are most pertinent to users and system administrators are *SIGKILL* and *SIGTERM*. Always try *SIGTERM* first, because it stops a process gracefully, ensuring that any child processes are handed off to *INIT* and not orphaned, and parent processes are informed. The one downside to *SIGTERM* is the process can elect to ignore it.

Use *SIGKILL* only when *SIGTERM* is ineffective. *SIGKILL* cannot be ignored, and it also kills child processes, which may affect other processes. A process can be left in limbo as a zombie process because the parent is not informed. Zombie processes are no big deal by themselves, they just sit there not doing anything. You can see if you have any in the header of *top*, on the right side of the Tasks line. The following example shows two zombies:

```
Tasks: 249 total, 1 running, 248 sleeping, 0 stopped, 2 zombie
```

You don't need to do anything as the parent application should automatically clean them up. If it doesn't, it's not a big deal, unless it generates a horde of zombies. That tells you there is a problem with the application. You can't kill zombies because they are already dead. They use a miniscule bit of system resources, but if you want to try getting rid of them, try sending them a *SIGCHLD*:

```
$ sudo kill -s SIGCHLD 1299
```

If the same process keeps using excessive system resources, review the configuration file or settings of its program to see if there are errors, or if you can tune it to be more efficient. Check your logfiles (Recipe 20.1) for clues.

See Also

- *man 1 top*
- *man 1 kill*

20.9 Viewing Selected Processes in top

Problem

You want to track just one or a small number of processes.

Solution

Start *top* with a comma-delimited list of the processes you want to track:

```
$ top -p 4548, 8685, 9348
top - 10:57:39 up 44 min,  2 users,  load average: 0.10, 0.11, 0.21
Tasks:   3 total,   0 running,   3 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.2 us,  0.2 sy,  0.0 ni, 99.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :  15691.4 total,  12989.5 free,   1467.4 used,   1234.4 buff/cache
MiB Swap:  15258.0 total,  15258.0 free,      0.0 used.  13601.1 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   2907 mysql     20   0 1775688  78584  18396 S   0.0   0.5   0:00.22 mysqld
    927 root      20   0 1569764  39072  29320 S   0.0   0.2   0:00.16 libvirtd
    822 root      20   0   11040   6384   4732 S   0.0   0.0   0:00.02 smartd
```

Now you can keep an eye on what you want to see, without having to wade through the remaining hordes of processes. Press the equals sign key (=) to return to the complete process list.

See Also

- *man 1 top*
- *man 1 kill*

20.10 Escaping from a Frozen Graphical Desktop

Problem

There you were, working happily, when your graphical desktop froze. The cursor moves, but very slowly, or it does not move at all.

Solution

This is one of my favorite Linux features: dropping to the console from a graphical session. Press Ctrl-Alt-F2, and you should find yourself at the plain-text console that lies underneath your graphical session (Figure 20-5).



Figure 20-5. The Linux console

Log in, and now you can run some troubleshooting commands. Start with *top* to find the wayward process clogging your system and kill it, check log files, run other diagnostics, whatever you need to do. Once you have resolved the problem, press Alt-F7 to return to your graphical desktop. The worse case is you will have to shut down or reboot, which is better than a forced shutdown from pressing the power button.

The various Linuxes map these key combinations in different ways. Alt-F7 is traditional for the graphical session. Fedora uses Alt-F1. It hurts nothing to try them all.

Discussion

Another option is to open an SSH session from another computer and try to unfreeze your graphical desktop.

To me this is the best of all worlds, having both the console and graphical environment available at the same time.

A forced shutdown isn't necessarily a disaster as it used to be, especially when you are using a journaling filesystem such as Ext4, XFS, or Btrfs.

The standard configuration is seven consoles, F1 through F7. Each one is an independent login session.

Use Ctrl-Alt-F*n* to leave a graphical session and enter the console, and when you are in the console, use Alt-F*n*.

20.11 Troubleshooting Hardware

Problem

You think you have failing hardware and need to know how to test it.

Solution

When you suspect a hardware problem, first try the recipes in this chapter about hardware monitoring. Some system UEFI firmwares include hardware health monitors.

If the monitors do not point you in a clear direction, shut down the machine, open the case, clean out the dust, clean the filters if there are any, then remove and reseat everything that can be unplugged and reconnected: power cables, SATA cables, graphics adapters and other PCI expansion cards, memory modules, and fan connectors. Carefully reconnect everything, and pay special attention to reseating your memory modules correctly.



How to Not Kill Your Hardware, or Yourself

Be careful! Ground yourself by touching something else to discharge static electricity. Wear an anti-static wrist strap, and place your components on an anti-static mat. Unplug your machine, and NEVER touch anything inside the case while it is plugged in.

Test the power supply with a multimeter, if you know how to do this, or try a different power supply. Testing with a multimeter is fairly easy, and there are plenty of how-tos. If you have spare parts, swapping out a suspect component and trying a different one can pinpoint faulty hardware.

After you are finished and everything is back together, see if the problem is corrected. In my computer adventures a number of problems were solved by reseating the memory modules or moving them to different slots. Note that on most motherboards you must install RAM pairs in certain slots. Some issues related to RAM are data corruption, incomplete boots, and odd behaviors like when you press the power

switch to start up your system, it fluctuates like the power supply is faulty, and does not start.

Make certain that your case fans are oriented the right way. Air must be pulled into the case, usually from the front and sides, and then evacuated out the back.

There are quite a few hardware testers in Linux-land. Some vendors provide their own hardware and system testers; for example, Lenovo ThinkPads come with comprehensive testers that test every component on your system.

GtkStressTesting is a good utility for stress testing CPU, memory, and other components, and it extracts detailed motherboard information. Follow the instructions in the Setup Guide to install it on your system. It includes monitors similar to *lm-sensors*.

One feature it does not have is I/O monitoring, which you need to spot performance bottlenecks. For this, use *iotop*, which monitors disk performance in a *top*-like interface.

Discussion

It can be difficult to determine whether a problem is software or hardware. Be systematic and thorough, as hurrying takes longer. Use the available help for your Linux distribution, as there are always issues particular to each distro. Always read the release notes.

See Also

- *man 8 iotop*
- GtkStressTesting
- The documentation for your hardware components
- Your Linux distribution's documentation, forums, wikis, and release notes
- Chapter 10
- Recipe 20.6

CHAPTER 21

Troubleshooting Networks

Figuring out networking problems is just like any troubleshooting. Know your network, know how to use basic tools well, and be patient and systematic.

In this chapter we learn how to use *ping*, FPing, Nmap, *httping*, *arping*, and *mtr* to test connectivity, map networks, find rogue services, test website performance, find duplicate IP addresses, and find routing bottlenecks.

Diagnostic Hardware

If you find yourself stuck with mysterious unlabeled Ethernet and phone cabling, get yourself an Ethernet/telephone cable test and tone tracker. There are many that cost under $100. These come in two pieces: one emitter and one receiver. This goes fast with two people, one at each end of a cable. When you find both ends of a cable, label it and move on. You can do it alone, but it is faster with two people.

Multimeters are useful for a lot of jobs, such as finding shorts and opens, testing for continuity and attenuation, determining whether a wire is terminated correctly, testing electric outlets, and testing computer power supplies and motherboards. Adafruit is a great site to find excellent tutorials on using a multimeter and learning electronics.

Keep a few spare parts, if you can. Sometimes it is faster to swap out a network interface, cable, or switch to find a defective piece of hardware.

21.1 Testing Connectivity with ping

Problem

Some services or hosts on your network are not accessible or have intermittent failures. You want to figure out if it is a hardware problem, a problem with name services, routing, or something else.

Solution

When you are debugging network problems, start close, and systematically work from closer to farther. This means physical distance and how many routers there are to cross. Start within your local LAN segment. Then proceed to your next LAN segment, if you have more than one, crossing a single router. Then to the next one two routers away, and so on.

Start with good old *ping* to test connectivity. First, ping *localhost*:

```
$ ping localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.065 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.035 ms
```

Stop *ping* by pressing Ctrl-C. Pinging *localhost* first confirms that your network interface is up and operating. If you see "connect: Network is unreachable," there is a problem with your network interface. Keep some spare USB network interfaces on hand to quickly learn if you have a defective interface.

Once you have your network interface sorted, ping your hostname to test name resolution, and tell *ping* to stop after three pings:

```
$ ping -c 3 client4
PING client4 (192.168.1.97) 56(84) bytes of data.
64 bytes from client4 (192.168.1.97): icmp_seq=1 ttl=64 time=0.087 ms
64 bytes from client4 (192.168.1.97): icmp_seq=2 ttl=64 time=0.059 ms
64 bytes from client4 (192.168.1.97): icmp_seq=3 ttl=64 time=0.061 ms

--- client4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2046ms
rtt min/avg/max/mdev = 0.059/0.069/0.087/0.012 ms
```

If it returns the correct IP address, your name resolution is set up correctly. If it returns a localhost address, like 127.0.1.1, or "Name or service not known," something is haywire with your DNS configuration.

When your local DNS is fixed, ping one of your network hosts by hostname. If *ping* fails with "Destination Host Unreachable" try pinging its IP address. If that succeeds, check your DNS. If that fails with the same message, your hostname and address are incorrect, or the host is down.

If you cannot reach any external IP addresses, your network interface is probably healthy and the problem is upstream: your Ethernet cable, wireless access point, or switch. "Network is unreachable" means your machine is not connected to the network.

When you're hunting down the source of intermittent outages, set *ping* to run for a length of time, like 500 pings, spaced 2 seconds apart, so you don't overwhelm the host or your network, and output the results to a text file. The following example appends added information to the file, so you can stop and restart:

```
$ ping -c 500 -i 2 server2 >> server2-ping.txt
```

Or, use *tee* to see the output and record it in a file:

```
$ ping -c 500 -i 2 server2 | tee server2-ping.txt
```

On a multihomed host, use *ping -i interface-name* to specify which interface to use.

Discussion

Don't block *echo-request*, *echo-reply*, *time-exceeded*, or *destination-unreachable* ping messages. Some admins block all ping messages at their firewalls, and this is a mistake because many network functions require at least these four ping messages to operate correctly.

The *ping* command actually pings if you use the *-a* (audible) switch, though you will probably have to do a bit of setup to make it work. In olden times we had PC speakers built into computer cases, connected directly to the motherboard, and the kernel module that activated the case speaker automatically loaded at boot. You are probably familiar with the low-fi annoyings beeps emitted from this speaker, and perhaps you even ran some hacks to make it play music.

Now, in these here modern times, case speakers are mostly gone, and laptops mostly do not have a motherboard beep anymore. But most PC motherboards still support them, and the modern beep speaker is a little thing (Figure 21-1). You will probably have to buy one.

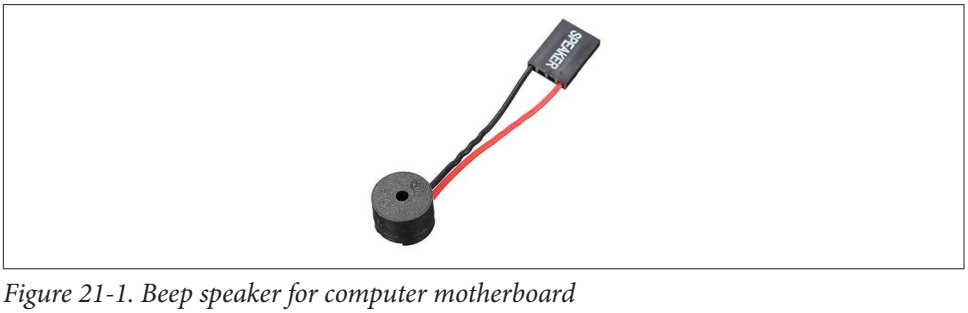


Figure 21-1. Beep speaker for computer motherboard

Once you have your beep speaker, load the *pcspkr* kernel module, then confirm it is loaded:

```
$ sudo modprobe pcspkr
$ lsmod|grep pcspkr
pcspkr                 16384  0
```

Now try it out. Drop to a plain console with Ctrl-Alt-F2, or fire up an X terminal, and use the *echo* command to play the ASCII bell character. All examples are the same thing, different representations of the ASCII character code 7:

```
$ echo -e "\a"
$ tput bel
$ echo -e '\007'
```

Or press Ctrl-G.

If you hear nothing in your graphical terminal, check its settings to enable sounds. *xfce4-terminal* and *gnome-terminal* both play the ASCII bell. *Konsole* supports using your choice of sound files for notifications, but it does not support the beep speaker.

See Also

- *man 8 ping*
- IANA list of ICMP parameters

21.2 Profiling Your Network with fping and nmap

Problem

You want to generate a list of all hosts and IP addresses on your network and probe for MAC addresses and open ports.

Solution

Use *fping* and *nmap* to probe your LAN, and record the results.

fping pings all the addresses in a range in sequence. This example pings a subnet once, reports which hosts are alive, queries DNS for the hostnames, and prints a summary:

```
$ fping -c1 -gAds 192.168.1.0/24 2>1 | egrep -v "ICMP|xmt" >> fping.txt
client1.net (192.168.1.15)     : [0], 84 bytes, 3.12 ms (3.12 avg, 0% loss)
server2.net (192.168.1.91)     : [0], 84 bytes, 5.34 ms (5.34 avg, 0% loss)
client4.net (192.168.1.97)     : [0], 84 bytes, 0.03 ms (0.03 avg, 0% loss)

     254 targets
       3 alive
```

```
     251 unreachable
       0 unknown addresses

     251 timeouts (waiting for response)

 0.03 ms (min round trip time)
 2.83 ms (avg round trip time)
 5.34 ms (max round trip time)
        3.575 sec (elapsed real time)
```

To see the unfiltered output, omit the *2>1 | egrep -v "ICMP|xmt"* part. Any offline machines will not be found, so you could run this at different times to try to capture everything. *>> fping.txt* appends the new results for each run.

This *nmap* example performs a similar task, with less verbose output:

```
$ sudo nmap -sn 192.168.1.0/24 > nmap.txt
Starting Nmap 7.70 ( https://nmap.org ) at 2021-03-31 18:30 PDT
Nmap scan report for client1.net (192.168.1.15)
Host is up (0.0052s latency).
MAC Address: 44:A5:6E:D7:8F:B9 (Unknown)
Nmap scan report for BRW7440BBC7CA75.net (192.168.1.39)
Host is up (1.0s latency).
MAC Address: 74:40:BB:C7:CA:75 (Unknown)
Nmap scan report for client4.net (192.168.1.97)
Host is up (0.47s latency).
MAC Address: 9C:EF:D5:FE:8F:20 (Panda Wireless)
Nmap scan report for server2.net (192.168.1.91)
Host is up.
Nmap done: 256 IP addresses (6 hosts up) scanned in 15.19 seconds
```

That is a rather indigestible lump, so insert a newline before each host and store the output in a new file:

```
$ awk '/Nmap/{print ""}1' nmap.txt > nmap2.txt
```

Now you have nice groupings:

```
Nmap scan report for client1.net (192.168.1.15)
Host is up (0.0052s latency).
MAC Address: 44:A5:6E:D7:8F:B9 (Unknown)

Nmap scan report for BRW7440BBC7CA75.net (192.168.1.39)
Host is up (1.0s latency).
MAC Address: 74:40:BB:C7:CA:75 (Unknown)

Nmap scan report for client4.net (192.168.1.97)
Host is up (0.47s latency).
MAC Address: 9C:EF:D5:FE:8F:20 (Panda Wireless)

Nmap scan report for server2.net (192.168.1.91)
Host is up.

Nmap done: 256 IP addresses (6 hosts up) scanned in 15.19 seconds
```

Probe the hosts on your network for open ports:

```
$ sudo nmap -sS  192.168.1.*
Starting Nmap 7.70 ( https://nmap.org ) at 2021-03-31 19:36 PDT
Nmap scan report for client2.net (192.168.1.15)
Host is up (0.027s latency).
Not shown: 997 closed ports
PORT     STATE    SERVICE
53/tcp   open     domain
80/tcp   open     http
MAC Address: 44:A5:6E:D7:8F:B9 (Unknown)

Nmap scan report for 192.168.1.39
Host is up (0.074s latency).
Not shown: 994 closed ports
PORT     STATE SERVICE
25/tcp   open  smtp
80/tcp   open  http
443/tcp  open  https
515/tcp  open  printer
631/tcp  open  ipp
9100/tcp open  jetdirect
MAC Address: 74:40:BB:C7:CA:75 (Unknown)
[...]
```

client2.net is running a DNS and web server. You can run the same probe from outside your firewall to see if they are visible outside of your network.

The second entry is interesting because it is a network printer running a whole mob of services. The printer documentation says they all have a purpose. The printer supports remote administration through a web control panel, so they could be disabled, if necessary.

Collect a list of hosts and their IP addresses:

```
$ nmap -sn 192.168.43.0/24 | grep 'Nmap scan report for' |cut -d' ' -f5,6
server2 (192.168.43.15)
dns-server (192.168.43.74)
client4 (192.168.43.14)
```

Discussion

nmap has numerous options for probing networks. Do not probe other people's networks without permission because it could be seen as a hostile act, probing for vulnerabilities.

Running a port scan takes some time, but it is a good idea to do this regularly to see what is happening on your network. It is basic security to run only necessary services and to disable everything else.

See Also

- *man 1 nmap*
- *https://nmap.org*
- *man 8 fping*
- *https://fping.org*

21.3 Finding Duplicate IP Addresses with arping

Problem

You want to search your network for duplicate IP addresses.

Solution

This example searches your network for 192.168.1.91 and sends four pings:

```
$ sudo arping -I wlan2 -c 4 192.168.1.91
ARPING 192.168.1.91
42 bytes from 9c:ef:d5:fe:01:7c (192.168.1.91): index=0 time=49.463 msec
42 bytes from 9c:ef:d5:fe:01:7c (192.168.1.91): index=1 time=458.306 msec
42 bytes from 9c:ef:d5:fe:01:7c (192.168.1.91): index=2 time=73.938 msec
42 bytes from 9c:ef:d5:fe:01:7c (192.168.1.91): index=3 time=504.482 msec

--- 192.168.1.91 statistics ---
4 packets transmitted, 4 packets received,   0% unanswered (0 extra)
rtt min/avg/max/std-dev = 49.463/271.547/504.482/210.659 ms
```

All of the MAC addresses are the same, so it found no duplicates. This is an example of *arping* finding duplicate IP addresses:

```
$ sudo arping -I wlan2 -c 4 192.168.1.91
ARPING 192.168.1.91
42 bytes from 9c:ef:d5:fe:01:7c (192.168.1.91): index=0 time=49.463 msec
42 bytes from 2F:EF:D5:FE:8F:20 (192.168.1.91): index=1 time=458.306 msec
42 bytes from 9c:ef:d5:fe:01:7c (192.168.1.91): index=2 time=73.938 msec
42 bytes from 2F:EF:D5:FE:8F:20 (192.168.1.91): index=3 time=504.482 msec
[...]

--- 192.168.1.91 statistics ---
4 packets transmitted, 4 packets received,   0% unanswered (0 extra)
rtt min/avg/max/std-dev = 49.463/271.547/504.482/210.659 ms
```

Use *nmap* to identify the two machines with the same IP address:

```
$ nmap -sn 192.168.43.0/24 | grep 'Nmap scan report for' |cut -d' ' -f5,6
```

Discussion

arp is the Address Resolution Protocol, matching IP addresses to MAC addresses.

An advantage of using DHCP to dynamically assign IP addresses is less risk of creating duplicates than setting static IP addresses manually. You can assign static addresses with DHCP; see Chapter 16.

arping is useful to see if a host is up when *ping* does not find it. Some folks like to block *ping*, which is not a good thing to do because it is essential to network functionality. *arping* cannot be blocked without disabling the ability for network hosts to communicate with each other. *arp*, the Address Resolution Protocol, maintains a table of MAC addresses. When a network host sends a packet to another host, *arp* matches the IP address to the MAC address, and then the packet can be delivered.

You can see what it looks like when *arp* probes your network to update its address table, with a packet sniffer like *tcpdump*:

```
$ sudo tcpdump -pi eth1 arp
listening on eth1, link-type EN1000MB (Ethernet), capture size 262144 bytes
21:19:36.921293 ARP, Request who-has client4.net tell m1login.net, length 28
21:19:36.921309 ARP, Reply client4.net is-at 9c:ef:d5:fe:8f:20
```

See Also

- Chapter 16
- *man 8 arping*

21.4 Testing HTTP Throughput and Latency with httping

Problem

You want to test a website that you host to see if it loads in a reasonable length of time.

Solution

httping measures HTTP server throughput and latency. Its simplest invocation tests latency:

```
$ httping -c4 -l -g www.oreilly.com
PING www.oreilly.com:443 (/):
connected to 184.86.29.153:443 (453 bytes), seq=0 time=292.25 ms
connected to 184.86.29.153:443 (453 bytes), seq=1 time=726.35 ms
connected to 184.86.29.153:443 (452 bytes), seq=2 time=629.11 ms
connected to 184.86.29.153:443 (453 bytes), seq=3 time=529.95 ms
--- https://www.oreilly.com/ ping statistics ---
```

```
4 connects, 4 ok, 0.00% failed, time 6179ms
round-trip min/avg/max = 292.2/544.4/726.3 ms
```

This doesn't tell you how long it takes pages to load, only how long it takes the server, in milliseconds, to respond to a HEAD request, which fetches only the page headers without the content. A GET (*-G*) request fetches the whole page:

```
$ httping -c4 -l -Gg www.oreilly.com
PING www.oreilly.com:443 (/):
connected to 104.112.183.230:443 (453 bytes), seq=0 time=2125.72 ms
connected to 104.112.183.230:443 (453 bytes), seq=1 time=701.94 ms
connected to 104.112.183.230:443 (453 bytes), seq=2 time=470.66 ms
connected to 104.112.183.230:443 (453 bytes), seq=3 time=433.11 ms
--- https://www.oreilly.com/ ping statistics ---
4 connects, 4 ok, 0.00% failed, time 7733ms
round-trip min/avg/max = 433.1/932.9/2125.7 ms
```

Add the *-r* switch to minimize DNS latency by resolving the hostname just once:

```
$ httping -c4 -l -rGg www.oreilly.com
PING www.oreilly.com:443 (/):
connected to 23.10.2.218:443 (452 bytes), seq=0 time=961.29 ms
connected to 23.10.2.218:443 (452 bytes), seq=1 time=1091.16 ms
connected to 23.10.2.218:443 (452 bytes), seq=2 time=925.46 ms
connected to 23.10.2.218:443 (452 bytes), seq=3 time=913.26 ms
--- https://www.oreilly.com/ ping statistics ---
4 connects, 4 ok, 0.00% failed, time 7894ms
round-trip min/avg/max = 913.3/972.8/1091.2 ms
```

If minimizing DNS latency makes a large difference, then you need to take a look at your nameservers.

Test an alternate port, such as 8080, by appending it to the URL:

```
$ httping -c4 -l -rGg www.oreilly.com:8080
```

Use the *-s* switch to display return codes, such as 200 OK, which indicates a successful page load:

```
$ httping -c4 -l -srGg www.oreilly.com
PING www.oreilly.com:443 (/):
connected to 23.10.2.218:443 (452 bytes), seq=0 time=920.88 ms 200 OK
connected to 23.10.2.218:443 (452 bytes), seq=1 time=857.60 ms 200 OK
connected to 23.10.2.218:443 (452 bytes), seq=2 time=1246.69 ms 200 OK
connected to 23.10.2.218:443 (452 bytes), seq=3 time=1134.91 ms 200 OK
--- https://www.oreilly.com/ ping statistics ---
4 connects, 4 ok, 0.00% failed, time 8249ms
round-trip min/avg/max = 857.6/1040.0/1246.7 ms
```

Discussion

Run multiple tests at different times of day to gather data that is representative of what your users are experiencing.

httping is not a super-sophisticated tester that digs deeply into your site to identify bottlenecks. It is a quick, simple tool to give you an idea of your overall site performance, and to tell you if you need to dig deeper to diagnose performance problems.

See Also

- HTTP return codes
- *man 1 httping*
- httping

21.5 Using mtr to Find Troublesome Routers

Problem

There is a site you are trying to access, and it is very slow or unreachable.

Solution

Use *mtr* (My Traceroute) to see where your packets are going astray. This works better on networks that you control, because the internet is vast and routes change, but when you are having trouble reaching a site it will provide useful information.

Let's see what wandering path takes us to *carlaschroder.com*:

```
$ mtr -wo LSRABW carlaschroder.com
Start: 2021-03-31T09:54:17-0700
HOST: client4                                Loss%   Snt   Rcv   Avg  Best  Wrst
  1.|-- m1login.net                            0.0%    10    10  55.5   1.2 199.6
  2.|-- 172.26.96.169                          0.0%    10    10  92.3  29.0 243.6
  3.|-- 172.18.84.60                           0.0%    10    10  84.5  29.3 220.3
  4.|-- 12.249.2.25                            0.0%    10    10  80.7  36.4 215.5
  5.|-- 12.122.146.97                          0.0%    10    10  65.6  34.8 156.6
  6.|-- 12.122.111.33                          0.0%    10    10  49.3  35.5  97.6
  7.|-- cr2.st6wa.ip.att.net                   0.0%    10    10  46.7  35.9  64.0
  8.|-- 12.122.111.109                         0.0%    10    10  57.9  31.4 215.4
  9.|-- 12.122.111.81                          0.0%    10    10  72.3  27.6 231.4
 10.|-- 12.249.133.242                         0.0%    10    10 101.2  31.7 263.1
 11.|-- ae6.cbs01.wb01.sea02.networklayer.com  0.0%    10    10  93.7  31.6 202.7
 12.|-- fc.11.6132.ip4.static.sl-reverse.com   0.0%    10    10 106.0  86.1 171.2
 13.|-- ae1.cbs02.eq01.dal03.networklayer.com 60.0%    10     4 102.0  86.5 115.8
 14.|-- ae0.dar01.dal13.networklayer.com       0.0%    10    10 103.7  80.3 230.8
 15.|-- 85.76.30a9.ip4.static.sl-reverse.com   0.0%    10    10 114.8  82.8 305.7
```

```
16.|-- a1.76.30a9.ip4.static.sl-reverse.com   0.0%    10    10 122.7  83.7 278.4
17.|-- hs17.name.tools                       0.0%    10    10 145.9  74.9 277.2
```

m1login.net is my network's internet gateway router. After that it is all the wild internet. Hop 13 could be a chokepoint, with 60% packet loss. Hop 13 could be part of a load-balancing cluster; note that hop 11 and hop 14 have the same domain name. If it is part of a cluster then the packet loss is not significant.

Ping the last hop, hs17.name.tools. The following example looks like everything is working fine:

```
$ ping -c 3 hs17.name.tools
PING hs17.name.tools (169.61.1.230) 56(84) bytes of data.
64 bytes from hs17.name.tools (169.61.1.230): icmp_seq=1 ttl=46 time=319 ms
64 bytes from hs17.name.tools (169.61.1.230): icmp_seq=2 ttl=46 time=168 ms
64 bytes from hs17.name.tools (169.61.1.230): icmp_seq=3 ttl=46 time=166 ms
[...]
```

If *mtr* reveals a problem, use *whois* to look up the domain owner and their contact information:

```
$ whois -H networklayer.com
```

whois also works for IP addresses. The *-H* switch disables the annoying legalese.

Capture *mtr* output in a file, with the date and time at the end of each entry:

```
$ mtr -r -c25 oreilly.com >> mtr.txt && date >> mtr.txt
```

Collect data over time by creating a cron job (Recipe 3.7) to run the preceding *mtr* once per hour, and let it run for a day or two. Don't forget to turn it off.

Discussion

mtr -wo LSRABW limits the number of columns to make the example fit better on this page. *mtr -w* is wide format for reports.

Save your records in case you need to report a problem; the *whois* examples show how to find who to contact.

mtr generates a lot of traffic, so take care to not run it too frequently.

See Also

- *man 8 mtr*

APPENDIX

Software Management Cheatsheets

Software on Linux comes in *packages*. These packages contain all the files that belong to a particular application, such as a web browser, word processor, and games. Linux systems use shared libraries, which are shared by multiple applications. Most packages on Linux are not self-contained, but depend on shared files.

The graphical software manager on most Linux distributions is GNOME-Software, also called Software (Figure A-1). Software is well organized, with categories and good search capabilities.

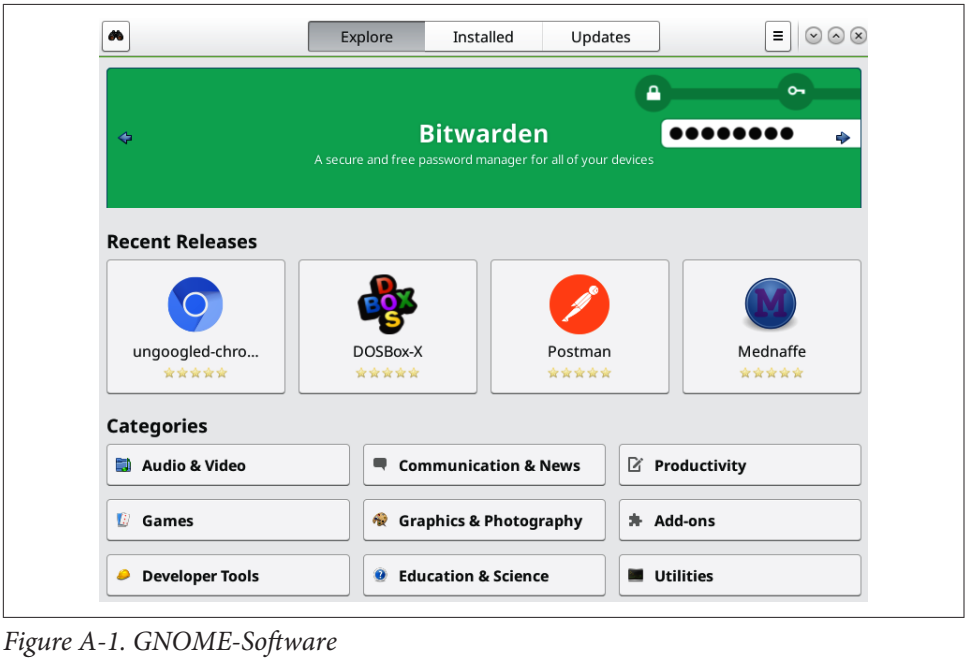


Figure A-1. GNOME-Software

Package Management Commands

Every Linux distribution uses three types of software management commands:

- A package manager, which manages only single packages. Fedora and openSUSE use the *rpm* package manager, Ubuntu uses *dpkg*.
- A dependency-resolving package manager. Fedora uses *dnf*, openSUSE uses *zypper*, and Ubuntu has *apt*. Dependency-resolving package managers ensure that any dependencies for a particular package are automatically resolved. For example, the gedit text editor has a long list of dependencies, as this example for *apt* illustrates:

```
$  apt depends gedit
gedit
  Depends: gedit-common (<< 3.37)
  Depends: gedit-common (>= 3.36)
  Depends: gir1.2-glib-2.0
  Depends: gir1.2-gtk-3.0 (>= 3.21.3)
  Depends: gir1.2-gtksource-4
  Depends: gir1.2-pango-1.0
  Depends: gir1.2-peas-1.0
  Depends: gsettings-desktop-schemas
  Depends: iso-codes
[...]
```

  Managing dependencies manually is difficult; dependency-resolving package managers make life many times easier for Linux users.

- Commands to manage groups of related packages, such as a graphical desktop, sound and video, or server stacks. openSUSE calls these *patterns*. Fedora calls them package groups. Ubuntu calls them *tasks*. The following example shows some openSUSE patterns:

```
$ zypper search --type pattern
S  | Name                | Summary                          | Type
---+---------------------+----------------------------------+--
[...]
   | mail_server         | Mail and News Server             | pattern
   | mate                | MATE Desktop Environment         | pattern
i+ | multimedia          | Multimedia                       | pattern
   | network_admin       | Network Administration           | pattern
   | non_oss             | Misc. Proprietary Packages       | pattern
   | office              | Office Software                  | pattern
   | print_server        | Print Server                     | pattern
[...]
```

Software packages are distributed from *repositories*, public servers that we download packages from. You can browse them online:

- Fedora Repositories
- openSUSE Repositories
- Ubuntu Packages Search

Every Linux distribution has official repositories, and then there is a whole world of third-party repositories. This appendix covers the basic commands for managing software and repository management on your Linux system.

Managing Software on Ubuntu

In this book, Ubuntu Linux stands in for a whole family of Debian-based distributions. Debian was first, then came hundreds of derivatives. The major Debian offspring use the same package management system, and the commands in this appendix should work the same way on all of them.

The three software management commands in this appendix are *dpkg*, *apt*, and *tasksel*.

Using add-apt to Install and Remove Repositories

When you add a software repository, you need the code name of your Ubuntu release. Get it with the following command:

```
$ lsb_release -sc
focal
```

You need the exact URL of the repository, which should be provided by the repository maintainers:

```
$ sudo add-apt-repository "deb http://us.archive.ubuntu.com/ubuntu/ focal \
universe multiverse"
```

Remove a repository:

```
$ sudo add-apt-repository -r "deb http://us.archive.ubuntu.com/ubuntu/ focal \
universe multiverse"
```

When you install or remove a repository, update your package cache:

```
$ sudo apt update
```

Run this command regularly to download repository updates, then install the updates:

```
$ sudo apt upgrade
```

Using dpkg to Install, Remove, and Inspect Packages

Remember from "Package Management Commands" on page 494 that *dpkg* only operates on single packages and does not resolve dependencies.

Install a package:

```
$ sudo dpkg -i packagename
```

Remove a package (does not remove configuration files):

```
$ sudo dpkg -r packagename
```

Remove a package and its configuration files:

```
$ sudo dpkg --purge packagename
```

List the contents of a package:

```
$ dpkg -L packagename
```

List all installed packages:

```
$ dpkg-query --listdpkg
```

Using apt to Search, Inspect, Install, and Remove Packages

apt is a dependency-resolving package manager, your everyday software manager command.

Search for a package:

```
$ apt search packagename
```

Limit the search to package names that include your search term:

```
$ apt search packagename --names-only
```

Get detailed information on a package:

```
$ apt show packagename
```

Install a package:

```
$ sudo apt install packagename
```

Remove a package (does not remove configuration files):

```
$ sudo apt remove packagename
```

Remove a package and its configuration files:

```
$ sudo apt remove purge packagename
```

Using tasksel

tasksel manages *tasks*, which are package groups.

List available tasks:

```
$ tasksel --list-tasks
```

Install a task:

```
$ sudo tasksel install task
```

Remove a task:

```
$ sudo tasksel remove task
```

Managing Software on Fedora

In this book, Fedora Linux represents a family of distributions based on Red Hat Linux. Red Hat, CentOS, Scientific Linux, Oracle Linux, and many others use the same package management system, and these commands should work on all of them.

The two software management commands in this chapter are *rpm* and *dnf*.

Using dnf to Manage Repositories

List all installed repositories, enabled and disabled:

```
$ dnf repolist --all
```

List enabled repositories:

```
$ dnf repolist --enabled
```

Show detailed information on enabled repositories:

```
$ dnf repolist --enabled
```

Add a repository:

```
$ sudo dnf config-manager --add-repo /etc/yum.repos.d/fedora_extras.repo
```

Enable the repository:

```
$ sudo dnf config-manager --set-enabled fedora-extras
```

Disable the repository:

```
$ sudo dnf config-manager --set-disabled fedora-extras
```

Using dnf to Manage Software

Search for a package:

```
$ dnf search packagename
```

Install a package:

```
$ sudo dnf install packagename
```

Remove a package:

```
$ sudo dnf remove packagename
```

Get information about a package:

```
$ dnf info packagename
```

Install updates:

```
$ sudo dnf upgrade
```

Get a list of package groups:

```
$ dnf grouplist
```

Install a package group:

```
$ sudo dnf groupinstall "package-group"
```

Remove a package group:

```
$ sudo dnf groupremove "package-group"
```

Using rpm to Install and Remove Packages

Install a package:

```
$ sudo rpm -i package
```

Upgrade a package:

```
$ sudo rpm -U package
```

Remove a package:

```
$ sudo rpm -e package
```

Using rpm to Get Information About Packages

List all files in an installed *rpm*:

```
$ rpm -ql package
```

Get complete information about an installed package:

```
$ rpm -qi package
```

See the changelog for a package:

```
$ rpm -q --changes package
```

Managing Software on openSUSE

openSUSE uses the RPM package format, like Fedora, but has a different dependency-resolving package manager, *zypper*.

Using zypper to Manage Repositories

List all installed repositories:

```
$ zypper repos
```

List installed repositories and show their URLs:

```
$ zypper repos -d
```

Enable a repository:

```
$ sudo zypper modifyrepo -e repo
```

Disable a repository:

```
$ sudo zypper modifyrepo -d repo
```

Add a new repository:

```
$ sudo zypper adderepo -name "MyNewRepoName" \
http://download.opensuse.org/distribution/leap/15.3/repo/oss/
```

Remove a repository:

```
$ sudo zypper removerepo MyNewRepoName
```

Download repository updates:

```
$ sudo zypper refresh
```

Using zypper to Manage Software

Update the system (run *sudo zypper refresh* first):

```
$ sudo zypper update
```

Search for a package (inexact search):

```
$ zypper search packagename
```

Search for a package (exact search):

```
$ zypper search -x packagename
```

Install a package:

```
$ sudo zypper install packagename
```

Remove a package:

```
$ sudo zypper remove packagename
```

List all software patterns:

```
$ sudo zypper -t patterns
```

Install a pattern:

```
$ sudo zypper -t pattern pattern-name
```

Index

Symbols

/ (root), partition for, 21
/ (slash), in copying directories, 167
/boot, partition for, 21, 186
/boot/grub/, 43, 161
/dev
  backups, 161
  disk names, 186
/etc
  backups, 161
  directory permissions, 132
  restoring, 163
/etc/default/grub
  options, 44-48
  purpose of, 43
/etc/fstab, 162, 253-255
/etc/group, 99
/etc/grub.d/, 43
/etc/hosts file
  on Dnsmasq servers, 371
  name resolution with, 368-370
  purpose of, 367
  testing/blocking sites, 371-372
/etc/inittab, 67
/etc/localtime, 405
/etc/passwd, 99
/home
  backups, 161
  partition for, 21, 186
  purpose of, 99
  tilde (~) shortcut, 139, 167
/media
  backups, 162
  mountpoints in, 252
/mnt
  backups, 161
  mountpoints in, 252
  subdirectories in, 443
/opt, backups, 161
/proc, 82-83, 161
/proc/filesystems, 246
/root, backups, 161
/srv, backups, 161
/sys, 161
/tmp
  backups, 161
  partition for, 21, 186
/usr/share/zoneinfo, 405
/var
  backups, 161
  partition for, 21, 186
/var/log, 458
@ (at sign), parameterized unit files, 315
~ (tilde), for /home directory, 139, 167

A

absolute filepaths, 136-137
ACPI (Advanced Configuration and Power Interface), 74
add-apt command, 495
addgroup command, 100, 115-116
Address Resolution Protocol (ARP), 488
adduser command, 100
  human users, creating, 113-114
  system users, creating, 114-115
advertising services over DHCP, 383-384
AGP (Accelerated Graphics Port), 230
aliasing commands, 124

allowing ports, 343-344
apt command, 496
Arch Linux, 431
ARP (Address Resolution Protocol), 488
arping command, 487-488
at sign (@), parameterized unit files, 315
attached journals (Ext4), finding, 259-260
authentication
  OpenSSH
    customizing Bash prompt, 292-293
    encryption algorithms, 273, 294-295
    host key generation, 276
    key fingerprints, 281
    multiple public keys, 284-285
    open session and run command, 290
    passphrase changes, 285-286
    password authentication, 279-281
    private key management with Keychain, 286-288
    public key authentication, 282-283
    purpose of, 273
    server configuration, 276-278
    server installation, 275
    sshfs command, 291-292
    syntax checking, 279
    tunneling X sessions, 288-290
    types of, 274
  on rsyncd servers, 180-182
  of sudo command, 128-129
authoritative name servers, 368
automated shutdowns (see scheduled shutdowns)
automated startups (see scheduled startups)
automatic backups
  with rsync command, 170
  with cp command, 164-165
automatic DHCP DNS entries, 386-388
automatic filesystem mounts, 253-255
awk command
  filtering lspci output, 232-233
  identifying kernel modules, 234

B

backgrounds for boot screen, customizing, 48-49
backups
  with cp command
    automatic, 164-165
    manual, 163
  devices for, 160
  importance of, 1
  with rsync command
    automatic, 170
    bandwidth limitations, 176
    excluding files, 170-171, 174-176
    including files, 172-174
    local backups, 166-168
    securing with SSH, 168-169
  with rsyncd servers
    access control, 180-182
    building server, 177-180
    MOTD files, 182-183
  selecting files, 161-162
bandwidth limitations with rsync command, 176
Bash prompt, customizing, 292-293
batch ownership, changing, 152-153
batch permissions, 150-151
batches of files, creating, 134-136
beep speakers, 483-484
binding zones (firewalld) to network interfaces, 336
BIOS/UEFI setup
  entering, 4-6
  GPT versus MBR, 187-188
  scheduled startups, 71-72
  Secure Boot, disabling, 3
  steps in startup, 38
  SystemRescue boot screens, 434-438
block devices, 12, 238
block zone (firewalld), 337
blocking
  IP addresses, 345-346
  ports, 343-344
  sites with /etc/hosts file, 371-372
blocks, 188-190
boot screen
  background, changing, 48-49
  customizing, 38
  displaying, 38, 40-41
  font colors, changing, 49-52
  grub rescue> prompt, 56-57
  grub> prompt, 54-56
  SystemRescue, 434-438
  themes, 52-53
bootable CD/DVD (see DVD installation media)
booting

from installation media, 2-3
to older kernels, 41-42
PXE booting, 75
Raspberry Pi into recovery mode, 420
steps in, 37-38
bootloaders, 37
(see also GRUB)
bootstrapping (see booting)
Bradley, David, 67
Broadcom SoC (system-on-a-chip), 409
Btrfs filesystems, 245, 269-271
bytes
decimal versus binary values, 190
in disk capacity, 188-190

C

CA (certificate authority), 298, 306
canceling timed shutdowns, 62
cd command, 137, 158
CD installation media (see DVD installation media)
centralized user management, 100
certificate authority (CA), 298, 306
certificates, 311
changing
default EasyRSA options, 311-312
default firewalld zones, 338
default target for zones, 346
default useradd settings, 106-107
file ownership, 151-153
passphrases, 285-286
printer configuration, 364-365
running system, 205
sudo password timeout, 126
SystemRescue, preserving changes, 453-454
character devices, 12
child processes, 84
chmod command
batch permissions, 150-151
octal notation
for directory permissions, 142-143
for file permissions, 140-142
special modes
removing, 146
setting, 143-145, 148-150
symbolic notation for file permissions, 146-148
chntpw command, 446-448
choosing (see selecting)
chown command, 151-153
Chromebooks, displaying hardware information, 241
chrony
advantages, 398
determining usage, 392-393
as NTP client, 397-398
as NTP server, 398-399
purpose of, 391
viewing statistics, 400-401
chronyc command, 397, 400-401
chroot environments, creating, 445-446
chroot jail, 179
clients
NTP
chrony configuration, 397-398
determining, 392-393
ntpd configuration, 401-402
timesyncd configuration, 394-395
OpenSSH, 275
OpenVPN
configuring and testing, 312-314
distributing configurations, 316-319
installing, 299-300
testing Dnsmasq from, 380-381
Cog System Info Viewer, 241
colors
for boot screen fonts, changing, 49-52
transparency, 51
command modes (parted), selecting, 191
commands, aliasing, 124
Common Unix Printing System (see CUPS)
composite video on Raspberry Pi, 417-419
configuration files (GRUB)
explained, 43-44
rebuilding, 40
reinstalling, 58
writing, 44-48
configuring
chrony
as NTP client, 397-398
as NTP server, 398-399
DHCP, 381-383
Dnsmasq for LAN DNS, 376-379
firewalld
backend, 332
for DNS and DHCP, 379
journal mode (Ext4), 257-259
journald, 461-462

logging in Dnsmasq, 388-389
networking, 323-324
ntpd
as NTP client, 401-402
as NTP server, 403-404
OpenSSH servers, 276-278
syntax checking, 279
OpenVPN, 312-314
smartmontools for email, 473-475
timesyncd, 394-395
wildcard domains in Dnsmasq, 389
connectivity testing
in OpenVPN, 300-302
with ping, 482-484
cooling
hardware, 456
Raspberry Pi, 413
Coordinated Universal Time (see UTC)
copying
as backup method
automatic copying, 164-165
manual copying, 163
files, 139-140
from failing hard disk, 448-449
over network, 442-445
partitions, 218-219
user files, 121
CoW (copy-on-write) filesystems, 245
cp command, 121, 139-140
automatic backups, 164-165
manual backups, 163
purpose of, 159
CPU information, listing, 238-240
cron
passphrase management with Keychain, 287
scheduled shutdowns, 69-71
crontab command
automatic backups, 164-165, 170
scheduled shutdowns, 69-71
Ctrl-Alt-Delete
configuration
in /etc/inittab file, 67
in graphical environments, 66
with systemd, 68-69
purpose of, 59
rebooting with, 66-67
CUPS (Common Unix Printing System), 347
driverless printing, 349, 357-359
"Forbidden" error message, 360-361
local printers
installing, 350-354
sharing, 359-360
naming printers, 354-355
network printers, installing, 355
printer configuration, changing, 364-365
printer drivers, 348
installing, 362-364
web interface for, 350
custom packages in Linux installs, 23-28
custom partitioning, installing Linux with, 18-21, 186, 197
customizing
Bash prompt, 292-293
boot screen, 38
background, 48-49
font colors, 49-52
themes, 52-53
EasyRSA, 311-312
firewalld zones, 339-340
well-known user directories, 108-110
cwd (current working directory), 137

D

daemons, 84
database backups, 162
date command, 405
dd command, 13-14, 413-415
dd-rescue command, 449
ddrescue command, 448-449
debugging (see troubleshooting)
default boot entry, setting, 45
default EasyRSA options, changing, 311-312
default firewalld zones
changing, 338
targets, changing, 346
default permissions, 153-154
default useradd settings, changing, 106-107
deleting
files/directories, 137-138
filesystems
with parted command, 250-251
while preserving partition, 213-214
groups with delgroup command, 120
partitions
with Gparted, 210-211
with parted command, 198-199, 250-251
recovering deleted, 199, 214
user files, 121

users
with deluser command, 119
with userdel command, 118-119
delgroup command, 120
deluser command, 119
dependency-resolving package managers, 494
desktop environments, installing multiple, 28
dhclient command, 382
DHCP (Domain Name Configuration Protocol)
advertising services, 383-384
automatic DNS entries, 386-388
configuring, 381-383
configuring firewalld for, 379
finding servers, 372-373
with Raspberry Pi, 428-429
static IP addresses, 385
subnet zones, 384-385
Diffie-Hellman, 309
dig command, 380-381
directories
backups (see backups)
copying/moving, 139-140
creating, 133-134
deleting, 137-138
filepaths, absolute/relative, 136-137
hiding, 157-158
ownership, changing, 151-153
permissions
in /etc, 132
default, 153-154
octal notation, 142-143
purpose of, 131
special modes, 143-145
renaming, 139-140
soft links, 154-157
well-known user, customizing, 108-110
disabled services
explained, 88
listing, 87
disabling
Ctrl-Alt-Delete, 68
firewalls, 440-441
repositories
with dnf command, 497
with zypper command, 499
Secure Boot, 3, 432
services, 92-93
user accounts, 117-118
disconnecting from network, 330
disks
blocks and sectors, 188-190
displaying existing, 192-194, 207-208
naming, 186
nonbootable, creating GPT partitions on, 195-197
partitions (see partitions)
terminology, 185, 206
displaying
boot screen, 38, 40-41
chrony statistics, 400-401
existing partitions/disks, 192-194
log files, 457-460
partitions, 207-208
processes, 477
startup messages, 79
UID/GID, 101-102
distro-hopping, 2
DistroWatch, 3, 7
dmesg command, 257, 457-458
dmz zone (firewalld), 337
dnf command, 497
DNS (Domain Name System)
automatic entries for DHCP clients, 386-388
configuring Dnsmasq for, 376-379
configuring firewalld for, 379
finding servers, 372-373
with Raspberry Pi, 428-429
server types, 367-368
Dnsmasq
/etc/hosts on, 371
configuring for LAN DNS, 376-379
DHCP
advertising services, 383-384
automatic DNS entries, 386-388
configuring, 381-383
static IP addresses, 385
subnet zones, 384-385
installing, 374-375
logging, 388-389
purpose of, 367-368
systemd-resolved and NetworkManager conflicts, 375-376
testing from client machine, 380-381
wildcard domains, 389
documentation, purpose of, 455
Documents directory, customizing, 108-110
Domain Name System (see DNS)

dot files, 157
  restoring, 163
downloading Linux, 3, 6
Downloads directory, customizing, 108-110
dpkg command, 496
driverless printing, 349, 357-359
drivers for printers, 348
  installing, 362-364
drop zone (firewalld), 337
drop-in files, 43
du command, 157, 202
dual-booting
  Linux/macOS, 2
  Linux/Windows, 1, 31-33
dumpe2fs command, 259-260
  reserved space settings, 263
duplicate IP addresses, finding, 487-488
DVD installation media, creating
  with K3b, 9-11
  for SystemRescue, 432
  with wodim, 12
Dynamic Host Configuration Protocol (see DHCP)
dynamic ports, 328

E

EasyRSA
  customizing, 311, 312
  installing, 304-305
  PKI, creating, 306-311
easyrsa init-pki command, 309
effective IDs, 102
email reports, configuring smartmontools for, 473-475
enabled services
  explained, 88
  listing, 87
enabling
  Ctrl-Alt-Delete, 68
  repositories
    with dnf command, 497
    with zypper command, 499
  services, 92-93
  SSH in SystemRescue, 440-441
encryption
  in OpenSSH, 273, 294-295
  in OpenVPN
    EasyRSA customization, 311-312
    EasyRSA installation, 304-305
    PKI creation, 306-311
    with static keys, 302-304
ephemeral ports, 328
Ethernet ports, adding to Raspberry Pi, 420-424
Ethernet/telephone cable test and tone trackers, 481
excluding files from backups, 170-171, 174-176
exFAT filesystems, 246
  creating, 266-267
exFAT FUSE utility, 246, 266
existing filesystems, listing, 248-249, 438-439
existing partitions/disks, displaying, 192-194, 207-208
Ext4 filesystems, 245
  creating, 256-257
  external journals, 260-262
  freeing reserved space, 262-263
  journal mode, configuring, 257-259
  journals, finding attached, 259-260
extended partitions, 212
extending sudo password timeout, 126
external journals (Ext4), 260-262
external zone (firewalld), 337

F

FAT16 filesystems, 245
  creating, 267-268
  Virtual FAT, 249
FAT32 filesystems, 245
  creating, 267-268
  Virtual FAT, 249
Fedora Linux
  custom package installation, 26-28
  rebuilding GRUB configuration files, 40
  software management commands, 497-498
  sudo authentication, 129
filepaths, absolute/relative, 136-137
files
  backups (see backups)
  blocks and sectors, 188-190
  copying
    with cp command, 139
    with cp command, 140
    from failing hard disk, 448-449
    over network, 442-445
  creating, 133-134
    in batches, 134-136
  deleting, 137-138
  excluding from backups, 170-171, 174-176

hiding, 157-158
including in backups, 172-174
ownership
changing, 151-153
types of, 131
permissions
in batches, 150-151
default, 153-154
octal notation, 140-142
purpose of, 131
special modes, 131, 143-145
symbolic notation, 146-148
types of, 131
printing to PDF, 365-366
renaming, 139-140
selecting
for backups, 161-162
for restores, 162-163
soft/hard links, 154-157
user files, finding/managing, 120-122
filesystems
Btrfs, creating, 269-271
CoW (copy-on-write), 245
creating with GParted, 220-220
deleting
with parted command, 250-251
while preserving partition, 213-214
exFAT, creating, 266-267
Ext4
configuring journal mode, 257-259
creating, 256-257
external journals, 260-262
finding attached journals, 259-260
freeing reserved space, 262-263
FAT16, creating, 267-268
FAT32, creating, 267-268
journaling, 245
labels, 205
Linux support for, 243
listing
existing, 248-249, 438-439
supported, 246-247
managing in SystemRescue, 450
mounting, 251-253, 443
automatically, 253-255
mountpoints, 244
operational overview, 244-246
primary, 212
remote, mounting with sshfs, 291-292
resizing, 200-202, 249
shrinking, 202-203
64-bit, 244
XFS
creating, 263
resizing, 264-265
filtering
lshw command output, 226
lspci command output, 231-233
filters for printing, 349
find command, 120-122
batch ownership, 152-153
batch permissions, 150-151
hard links, 156
finding
attached journals (Ext4), 259-260
DNS and DHCP servers, 372-373
duplicate IP addresses, 487-488
supported printers, 348
user files, 120-122
findmnt command, 253
firewall-applet interface, 425
firewall-cmd command, 331, 335
firewall-config interface, 326, 425
firewalld
configuring for DNS and DHCP, 379
installing, 330-331
internet sharing with Raspberry Pi, 424-427
IP addresses, blocking, 345-346
iptables/nftables, configuring as backend, 332
NetworkManager integration, 342-343
overview, 325-326
ports, allowing/blocking, 343-344
services, listing, 335-336
version, determining, 331
zones
changing default, 338
changing default target, 346
creating, 340-341
customizing, 339-340
listing, 332-334
predefined, 337-338
purpose of, 325-326
removing, 341
selecting/setting, 336-338
firewalls
determining active, 328-329
disabling, 440-441

firewalld (see firewalld)
operational overview, 326-327
on Raspberry Pi, 424-427
font colors for boot screen, changing, 49-52
"Forbidden" error message in CUPS, 360-361
forked processes, 84
fping command, 484
free space, displaying, 207-208
freeing reserved space in Ext4 filesystems, 262-263
frozen graphical environments, troubleshooting, 478-479
FUSE (Filesystem in Userspace), 246, 266

G

Gates, Bill, 67
GECOS data, 104
generated services, 88
GID (group identification)
displaying, 101-102
numbering range, 111
GMT (Greenwich Mean Time), 392
GNOME Disks, 267
GNOME-Software, 493
GParted (GNOME Partition Manager)
filesystems
creating, 220-220
deleting while preserving partition, 213-214
GPT partition table, creating, 209-210
partitions
copying, 218-219
creating, 211-212
deleting, 210-211
displaying, 207-208
moving, 216-218
recovering deleted, 214
resizing, 215-216
privileges, 206
purpose of, 205
GPT (GUID Partition Table), 186-188
creating new, 209-210
creating partitions on nonbootable disks, 195-197
primary filesystems, 212
graphical environments
Ctrl-Alt-Delete configuration, 66
CUPS web interface, 350
for lm-sensors, 467-469
for software management, 493
SystemRescue, 432
troubleshooting, 478-479
Greenwich Mean Time (GMT), 392
grep command, 457-458
grounding, 479
group identification (see GID)
groupadd command, 100, 110-111
groupdel command, 100
groups
assigning users, 112
commands for, 100
creating
with groupadd command, 110-111
with addgroup command, 115-116
deleting, 120
numbering range, 111
password files, checking integrity, 116-117
permissions, 132
primary, 104
privileges, purpose of, 99
sudo, 124-125
supplemental, 99, 104
grpck command, 116-117
GRUB (GRand Unified Bootloader), 37
boot screen
background, 48-49
customizing, 38
displaying, 38, 40-41
font colors, 49-52
grub rescue> prompt, 56-57
grub> prompt, 54-56
themes, 52-53
booting, steps in, 37-38
configuration files
explained, 43-44
rebuilding, 40
reinstalling, 58
writing, 44-48
legacy GRUB versus GRUB 2, 37
repairing from SystemRescue, 445-446
tab completion, 55
grub rescue> prompt, 56-57
grub> prompt, 54-56
GtkStressTesting, 480
GUID Partition Table (see GPT)

H

halt command, 59, 61

options, 63-64
halting with shutdown command, 62
hard disks
listing, 237-238
monitoring, 470-473
rescuing, 448-449
hard links, 154-157
hardening OpenVPN servers, 320-323
hardware
diagnostic hardware for network components, 481
identifying architecture, 240-241
listing
with hwinfo command, 227-228
with lsblk command, 237-238
with lscpu command, 238-240
with lshw command, 224-225
with lspci command, 228-230
with lsusb command, 235-236
preventing problems, 455-456
hard disk monitoring, 470-473
temperature monitoring, 465-467
testing, 479-480
hardware architectures
Linux, 4
Raspberry Pi, 409
hardware requirements for Raspberry Pi, 411-412
Hash-based Message Authentication Code (HMAC), 309
HDMI connections, Raspberry Pi video without, 417-419
headless Raspberry Pi, 427-428
hibernate mode, 65
hiding files/directories, 157-158
history of Raspberry Pi, 409-410
HMAC (Hash-based Message Authentication Code), 309
home zone (firewalld), 338
host keys
fingerprint retrieval, 281
generating, 276
password authentication, 279-281
purpose of, 274
hostname command, 370
HTTP throughput and latency testing, 488-490
httping command, 488-490
human users (see users)
hwinfo command
listing hardware information, 227-228
purpose of, 224
hybrid-sleep mode, 65

I

id command, 101-102
ifconfig command, 423
Imager, installing Raspberry Pi, 413-415
including files in backups, 172-174
indirect services, 88
init systems
determining availability, 82-83
legacy, 79, 80-81
runlevel management, 95-97
systemd (see systemd)
inodes, 156-157
installation media
booting from, 2-3
creating
with dd, 13-14
with K3b, 9-11
for SystemRescue, 432
with UNetbootin, 7-9
with wodim, 12
ISO format, 4
installing
Dnsmasq, 374-375
EasyRSA, 304-305
firewalld, 330-331
Linux, 1-2
basic installation, 15-18
custom package selection, 23-28
with custom partitioning, 18-21, 186, 197
in multiboot setup, 29-30
multiple desktop environments, 28
preserving partitions, 22-22
in Windows dual-boot setup, 31-33
local printers, 350-354
network printers, 355
OpenSSH server, 275
OpenVPN, 299-300
package groups, 498
packages
with apt command, 496
with dnf command, 498
with dpkg command, 496
with rpm command, 498
with zypper command, 499

printer drivers, 362-364
Raspberry Pi OS, 413-417
repositories
with add-apt command, 495
with dnf command, 497
with zypper command, 499
tasks, 497
themes, 52
integrity of password files, checking, 116-117
internal zone (firewalld), 338
internet-sharing firewall on Raspberry Pi, 424-427
iotop, 480
IP addresses
blocking, 345-346
finding duplicate, 487-488
ip command, 370
iptables, 326, 328
configuring as firewalld backend, 332
IPv4 masquerading, 427
ISO format, 4
ISO images, mounting, 35-36
iw command, 77-78

J

journal mode (Ext4), configuring, 257-259
journalctl command, 458-460
journald
configuring, 461-462
logging server creation, 462-464
journaling filesystems, 245
journals (Ext4)
external, 260-262
finding attached, 259-260

K

K3b, 9-11
kernel modules for PCI devices, identifying, 234
key fingerprints, retrieving, 281
Keychain, 274, 286-288
kill command, 94-95
killing processes, 94-95, 475-477

L

L1/L2/L3 CPU caches, 239
labels for partitions/filesystems, 205
latency, testing, 488-490
lexicographic ordering, 135-136
links to files/directories, 154-157
Linux
booting (see booting)
cost of, 2
downloading, 3, 6
dual-booting
with macOS, 2
with Windows, 1, 31-33
hardware architectures, 4
installing, 1-2
basic installation, 15-18
custom package selection, 23-28
with custom partitioning, 18-21, 186, 197
in multiboot setup, 29-30
multiple desktop environments, 28
preserving partitions, 22-22
in Windows dual-boot setup, 31-33
ISO images, mounting, 35-36
multibooting, 1, 29-30
recommendations for newbies, 3
size of distribution, 2
startup messages, displaying, 79
supported filesystems, 243
listing
filesystems
existing, 248-249, 438-439
supported, 246-247
firewalld services, 335-336
firewalld zones, 332-334
hardware
with hwinfo command, 227-228
with lsblk command, 237-238
with lscpu command, 238-240
with lshw command, 224-225
with lspci command, 228-230
with lsusb command, 235-236
package groups, 498
packages
with apt command, 496
with dnf command, 498
with dpkg command, 496
with rpm command, 498
repositories
with dnf command, 497
with zypper command, 499
services, 86-88
tasks, 497

live Linuxes, 1
lm-sensors
  graphical display for, 467-469
  temperature monitoring, 465-467
ln command, 154-157
local printers
  installing, 350-354
  sharing, 359-360
logging
  in Dnsmasq, 388-389
  journald configuration, 461-462
  logging server creation, 462-464
  purpose of, 455
  severity levels for, 460
  viewing log files, 457-460
logical partitions, 212
logrotate command, 389
loop devices, 35
loopback devices, 369-370
ls command, 135-136, 156
lsblk command, 13, 165
  listing existing filesystems, 248-249, 438-439
  listing hardware information, 237-238
  purpose of, 224
lscpu command
  listing CPU information, 238-240
  purpose of, 224
lshw command
  filtering output, 226
  listing hardware information, 224-225
  purpose of, 223
lspci command
  explanation of output, 230-231
  filtering output, 231-233
  identifying kernel modules, 234
  listing hardware information, 228-230
  purpose of, 224
lsusb command
  listing hardware information, 235-236
  purpose of, 224

M

macOS, dual-booting, 2
magic packets, 75
mailx, 473
manual backups with cp command, 163
manual time settings, 396
masked services
  explained, 88
  listing, 87
masking services, 92-93
MBR (Master Boot Record), 186-188
memory, CPU caches, 239
message of the day (MOTD) files on rsyncd servers, 182-183
mkdir command, 133
mkfs.exfat command, 266
mkfs.ext4 command, 256-257
mkfs.xfs command, 263
mkpart command, 267-268
modes (see permissions)
monitoring
  hard disks, 470-473
  temperature, 465-467
monitors, collecting hardware information, 227-228
MOTD (message of the day) files on rsyncd servers, 182-183
mount command, 253
mounting
  filesystems, 251-253, 443
    automatically, 253-255
  ISO images, 35-36
  remote filesystems with sshfs command, 291-292
mountpoint command, 253
mountpoints, 244
  creating, 251-253
moving
  files/directories, 139-140
  partitions, 216-218
  user files, 121
MS-DOS partition tables, 212
mtr command, 490-491
multibooting, 1, 29-30
multimeters, 481
multiple desktop environments, installing, 28
multiple Ethernet ports, adding to Raspberry Pi, 420-424
multiple public keys in OpenSSH, 284-285
Music directory, customizing, 108-110
mv command, 121, 139-140

N

name resolution, 367
  (see also Dnsmasq)
  with /etc/hosts file, 368-370
  DHCP

advertising services, 383-384
automatic DNS entries, 386-388
configuring, 381-383
configuring firewalld for, 379
finding servers, 372-373
static IP addresses, 385
subnet zones, 384-385
DNS
configuring Dnsmasq for, 376-379
configuring firewalld for, 379
finding servers, 372-373
server types, 367-368
Linux utilities for, 368
with Raspberry Pi, 428-429
naming
disks, 186
printers, 354-355
NAT (network address translation), 427
ncurses interface, 428
netfilter, 326
netstat command, 327
network address translation (NAT), 427
network printers, installing, 355
Network Time Protocol (see NTP)
networking
configuring, 323-324
copying files over, 442-445
disconnecting, 330
OpenVPN (see OpenVPN)
ports, numbering, 327-328
resources for information, 325
troubleshooting
connectivity tests, 482-484
diagnostic hardware, 481
duplicate IP addresses, 487-488
HTTP throughput and latency testing, 488-490
packet tracing, 490-491
port scanning, 484-486
NetworkManager, 326
configuring DHCP, 382-383
Dnsmasq conflicts, 375-376
firewalld integration, 342-343
firewalld zones, assigning, 338
importing .ovpn files, 318-319
name resolution, 368
nm-connection-editor, 388
OpenVPN installation, 300
nftables, 326, 328
configuring as firewalld backend, 332
nm-connection-editor, 388
nmap command, 372-373, 484-486
nmcli command, 330, 374, 381, 387
nobody users, 105, 115
nonbootable disks, creating GPT partitions on, 195-197
nonbooting system, troubleshooting
grub rescue> prompt, 56-57
grub> prompt, 54-56
nonnetworked printers, sharing, 359-360
NOOBS
booting into recovery mode, 420
installing Raspberry Pi, 415-417
nslookup command, 378, 389
NTP (Network Time Protocol)
clients
chrony configuration, 397-398
determining, 392-393
ntpd configuration, 401-402
timesyncd configuration, 394-395
purpose of, 391
servers
chrony configuration, 398-399
ntpd configuration, 403-404
ntpd
chrony advantages over, 398
determining usage, 392-393
as NTP client, 401-402
as NTP server, 403-404
purpose of, 391
ntpq command, 402

0

octal notation, 132
for directory permissions, 142-143
for file permissions, 140-142
special modes
removing, 146
setting, 143-145
OEM product key (Windows), recovering, 34
older kernels, booting to, 41-42
Open Virtual Private Network (see OpenVPN)
OpenSSH
authentication types, 274
Bash prompt, customizing, 292-293
clients, 275
enabling in SystemRescue, 440-441
encryption algorithms, 273, 294-295

host keys, generating, 276
key fingerprints, 281
open session and run command, 290
password authentication, 279-281
public key authentication, 282-283
multiple public keys, 284-285
passphrase changes, 285-286
private key management with Keychain, 286-288
purpose of, 273
remote filesystems, mounting with sshfs, 291-292
rsync backups, 168-169
server configuration, 276-278
server installation, 275
syntax checking, 279
tunneling X sessions, 288-290
utilities in, 273-274
openssl command, 311
openSUSE
creating Btrfs filesystem, 269-271
custom package installation, 23-28
rebuilding GRUB configuration files, 40
software management commands, 499-499
sudo authentication, 129
OpenVPN
client configurations, distributing, 316-319
configuring and testing, 312-314
connectivity testing, 300-302
encryption
EasyRSA customization, 311-312
EasyRSA installation, 304-305
PKI creation, 306-311
with static keys, 302-304
installing, 299-300
managing with systemctl, 315
networking, configuring, 323-324
overview, 297-298
server hardening, 320-323
openvpn command, 313
operating systems for Raspberry Pi, 408
installing, 413-417
optical disks (see DVD installation media)
.ovpn files, 316-319
ownership
changing, 151-153
types of, 131

P

package groups, 494, 498
package managers, 494
packages, 493
custom selection during install, 23-28
installing
with apt command, 496
with dnf command, 498
with dpkg command, 496
with rpm command, 498
with zypper command, 499
listing
with apt command, 496
with dnf command, 498
with dpkg command, 496
with rpm command, 498
management commands
in Fedora, 497-498
in openSUSE, 499-499
terminology, 494-495
in Ubuntu, 495-497
removing
with apt command, 496
with dnf command, 498
with dpkg command, 496
with rpm command, 498
with zypper command, 499
searching
with apt command, 496
with dnf command, 497
with zypper command, 499
upgrading with rpm command, 498
packet tracing, 490-491
parameterized unit files, 315
parent processes, 84
parted command
command modes, 191
filesystems
creating FAT16/FAT32, 267-268
deleting, 250-251
resizing XFS, 265
partitions
creating on nonbootable disks, 195-197
deleting, 198-199
displaying existing, 192-194
recovering deleted, 199
resizing, 200-202
shrinking, 202-203
purpose of, 185

partition tables, 186-188
  creating, 209-210
  MS-DOS, 212
partitions
  blocks and sectors, 188-190
  copying, 218-219
  creating, 211-212
    with Linux install, 18-21, 186, 197
    in SystemRescue, 451-452
  deleting
    with Gparted, 210-211
    with parted command, 198-199, 250-251
  displaying existing, 192-194, 207-208
  filesystems and, 244
  GPT, creating on nonbootable disks, 195-197
  labels, 205
  listing, 237-238
  managing in SystemRescue, 450
  moving, 216-218
  preserving
    in Linux installs, 22-22
    while deleting filesystem, 213-214
  purpose of, 186
  in Raspberry Pi, 415
  recovering deleted, 199, 214
  resizing, 200-202, 215-216, 249
  shrinking, 30, 202-203, 215-216
  unmounting, 190
passphrase-less authentication in OpenSSH, 274
passphrases
  changing, 285-286
  managing with Keychain, 286-288
passwd command, 100
  disabling accounts, 117-118
  privileges, 102
password authentication in OpenSSH, 274, 279-281
passwords
  checking file integrity, 116-117
  extending sudo timeout, 126
  for new users, 103
  root
    managing, 127
    resetting, 439-440
  Shadow Password Suite, 100
  for sudo command, 128-129
  Windows, resetting, 446-448
patience, importance of, 456-457
patterns, 171, 494
PCI devices
  collecting hardware information, 228-230
  kernel modules, 234
PCIe (PCI Express), 230
PDF, printing files to, 365-366
performance of external journals (Ext4), 260-262
permissions
  for directories
    in /etc, 132
    default, 153-154
    octal notation, 142-143
    purpose of, 131
    special modes, 143-145
  for files
    in batches, 150-151
    default, 153-154
    octal notation, 140-142
    purpose of, 131
    special modes, 131, 143-145
    symbolic notation, 146-148
    types of, 131
  of groups, 132
php command, 406
Pi (see Raspberry Pi)
Picture directory, customizing, 108-110
PID (process ID) 1, 83
ping command, 482-484
pinout command, 421, 424
PKI (public key infrastructure)
  creating, 306-311
  EasyRSA
    customizing, 311-312
    installing, 304-305
Poettering, Lennart, 82
pool servers, 392, 395
port scanning, 372-373, 484-486
ports
  allowing/blocking, 343-344
  numbering, 327-328
  sshd, 327
postfix, 473
PostScript Printer Description (PPD) files, 349
power conditioners, 456
power management
  ACPI sleep states, 74
  sleep modes, 64-65

poweroff command, 59, 60
  options, 63-64
  symlink with Ctrl-Alt-Delete, 68
PPD (PostScript Printer Description) files, 349
pre-installed Linux, 2
preserving
  partitions
    in Linux installs, 22-22
    while deleting filesystem, 213-214
  SystemRescue changes, 453-454
preventing hardware problems, 455-456
  hard disk monitoring, 470-473
  temperature monitoring, 465-467
primary filesystems, 212
primary groups, 104
printers
  configuration, changing, 364-365
  CUPS web interface, 350
  driverless printing, 349, 357-359
  drivers, 348
    installing, 362-364
  local
    installing, 350-354
    sharing, 359-360
  naming, 354-355
  network, installing, 355
  selecting, 347-348
  supported, finding, 348
printing
  files to PDF, 365-366
  troubleshooting, 360-361, 366
private keys, 310
  passphrases
    changing, 285-286
    managing with Keychain, 286-288
  purpose of, 274
private ports, 328
privileges, 132
  (see also permissions)
  for GParted, 206
  limitations, 132
  for passwd command, 102
  purpose of, 99
  root (see root privileges)
processes
  explained, 84-85
  killing, 94-95, 475-477
  viewing, 477
ps command, 83, 84, 392-393, 401
Psensor, 467-469
pstree command, 84-85
public key authentication in OpenSSH, 274, 282-283
  multiple public keys, 284-285
  passphrase changes, 285-286
  private key management with Keychain, 286-288
public key infrastructure (see PKI)
public keys, 274, 310
public zone (firewalld), 337
pwck command, 116-117
pwd command, 137
PXE booting, 75

R

RAID devices, collecting hardware information, 227-228
Raspberry Pi
  advantages/disadvantages, 407-408
  booting into recovery mode, 420
  cooling, 413
  hardware architecture, 409
  hardware requirements, 411-412
  history and purpose, 409-410
  internet-sharing firewall on, 424-427
  multiple Ethernet ports, 420-424
  name resolution with, 428-429
  operating systems, 408
    installing, 413-417
  products available, 409
  running headless, 427-428
  startup/shutdown, 410-411
  tutorials, 412
  video connections, 417-419
raspi-config, 427-428
real IDs, 101
real-time clock (see RTC)
reboot command, 59, 61
  options, 63-64
rebooting
  with Ctrl-Alt-Delete, 66-67
  to different system states, 95-97
  with halt command, 63
  with poweroff command, 63
  with reboot command, 63
  with shutdown command, 62
rebuilding GRUB configuration files, 40
recovering

deleted partitions, 199, 214
with SystemRescue (see SystemRescue)
Windows OEM product key, 34
recovery mode, booting Raspberry Pi, 420
recursive resolvers, 367
registered ports, 328
reinstalling GRUB configuration files, 58
relative filepaths, 136-137
remote access (see OpenSSH; OpenVPN)
remote filesystems, mounting with sshfs command, 291-292
remote startups
with Wake-on-LAN, 75-77
with WoWLAN, 77-78
removing
firewalld zones, 341
package groups, 498
packages
with apt command, 496
with dnf command, 498
with dpkg command, 496
with rpm command, 498
with zypper command, 499
repositories
with add-apt command, 495
with zypper command, 499
special modes, 146
renaming files/directories, 139-140
repositories, 495
disabling
with dnf command, 497
with zypper command, 499
enabling
with dnf command, 497
with zypper command, 499
installing
with add-apt command, 495
with dnf command, 497
with zypper command, 499
listing
with dnf command, 497
with zypper command, 499
removing
with add-apt command, 495
with zypper command, 499
rescue command, 199
rescuing hard disks, 448-449
reserved space in Ext4 filesystems, freeing, 262-263
reset vector, 38
resetting
root password, 439-440
Windows password, 446-448
resize command, 203
resizing
filesystems, 200-202, 249
partitions, 200-202, 215-216, 249
XFS filesystems, 264-265
resolvconf, 368
resolving Dnsmasq conflicts, 375-376
restoring files, what to restore, 162-163
restricted deletion bit, 145, 149
rich rules, 345-346
rm command, 121, 137-138
root name servers, 368
root password
managing, 127
resetting, 439-440
root privileges
for shutdown commands, 60
via shell escape, 124
root users, 99
abilities of, 132
changing sudo authentication, 128-129
changing sudo password timeout, 126
individual sudo configurations, 127
limiting powers with sudo, 123-126
switching to, 122-123
routers, packet tracing, 490-491
RPi (see Raspberry Pi)
rpm command, 498
rsync command
automatic backups, 170
bandwidth limitations, 176
excluding files, 170-171, 174-176
including files, 172-174
local backups, 166-168
purpose of, 159
secure backups with SSH, 168-169
rsyncd servers
access control, 180-182
building, 177-180
MOTD files, 182-183
RTC (real-time clock), 392
manual settings, 396
scheduled startups with, 73-75
rtcwake command, 73-75
runlevels (SysV), 95-97

running system, changing, 205

S

S.M.A.R.T. (Self-Monitoring Analysis and Reporting Technology), 470-473
saved IDs, 102
scheduled shutdowns
  with cron, 69-71
  purpose of, 59
scheduled startups
  with BIOS/UEFI setup, 71-72
  purpose of, 59
  with rtcwake command, 73-75
scp command, 273, 442-445
searching packages
  with apt command, 496
  with dnf command, 497
  with zypper command, 499
sectors, 188-190
Secure Boot, disabling, 3, 432
secure remote access (see OpenSSH; OpenVPN)
Secure Sockets Layer (SSL), 298
selecting
  command modes (parted), 191
  files
    for backups, 161-162
    for restores, 162-163
  firewalld zones, 336-338
  printers, 347-348
sensors command, 466
sensors-detect command, 465
servers
  logging, creating, 462-464
  OpenSSH
    configuring, 276-278
    installing, 275
  OpenVPN
    configuring and testing, 312-314
    hardening, 320-323
    installing, 299-300
    networking, configuring, 323-324
  rsyncd
    access control, 180-182
    building, 177-180
    MOTD files, 182-183
  time (see time servers)
services, 84
  advertising over DHCP, 383-384
  enabling/disabling, 92-93
  in firewalld, 326
    listing, 335-336
  killing, 94-95
  listing, 86-88
  masking/unmasking, 92-93
  querying status, 89-90
  starting/stopping, 91-92
  states, 87-88
  system users (see system users)
setgid mode, 132
  octal notation, 143-145
  symbolic notation, 148-150
setuid mode, 132
  octal notation, 143-145
  symbolic notation, 148-150
severity levels for system logs, 460
sftp command, 273
Shadow Password Suite, 100
shared libraries, 493
sharing local printers, 359-360
shell escape, 124
shortcuts to files/directories, 154-157
shrinking
  filesystems, 202-203
  partitions, 30, 202-203, 215-216
shutdown command, 59, 61
  options, 61-63
shutdown commands
  Ctrl-Alt-Delete as, 68
  halt command options, 63-64
  legacy commands
    purpose of, 59
    symlinks for, 60
  poweroff command options, 63-64
  reboot command options, 63-64
  root privileges, 60
  scheduled shutdowns
    with cron, 69-71
    purpose of, 59
  shutdown command options, 61-63
  systemctl, 60-61
SIGCHILD, 477
SIGKILL, 95, 476
signals, 476-477
SIGTERM, 95, 476
64-bit filesystems, 244
slash (/), in copying directories, 167
sleep modes, 64-65, 74

slots, 225
slow startups, troubleshooting, 98
sluggish systems, troubleshooting, 475-477
smartctl command, 470-471
smartd, configuring for email, 473-475
smartmontools
  configuring for email, 473-475
  hard disk monitoring, 470-473
soft links, 154-157
Software (graphical environment), 493
software management commands
  in Fedora, 497-498
  in openSUSE, 499-499
  terminology, 494-495
  in Ubuntu, 495-497
special modes
  octal notation
    removing, 146
    setting, 143-145
  symbolic notation, setting, 148-150
  types of, 131, 132
SquashFS, 434
ss command, 378
SSH (see OpenSSH)
ssh command, 273, 294-295
ssh-add command, 273
ssh-agent command, 274
ssh-copy-id command, 273, 282-283
ssh-keygen command, 273, 276
  multiple public keys, 284
  passphrase changes, 285-286
ssh-keyscan command, 273
sshd, 273
  ports, 327
sshfs command, 274, 291-292, 442-445
SSL (Secure Sockets Layer), 298
starting
  Raspberry Pi, 410-411
  services, 91-92
  slow startups, troubleshooting, 98
  SystemRescue, 432-434
startup commands
  remote startups
    with Wake-on-LAN, 75-77
    with WoWLAN, 77-78
  scheduled startups
    with BIOS/UEFI setup, 71-72
    purpose of, 59
    with rtcwake command, 73-75
startup messages, displaying, 79
startx command, 432
stat command, 69, 253
static IP addresses, assigning via DHCP, 385
static keys, 302-304
static services
  explained, 88
  listing, 87
statistics for chrony, displaying, 400-401
sticky bits, 132
  octal notation, 143-145
  symbolic notation, 148-150
stopping
  Raspberry Pi, 410-411
  services, 91-92
su command, 122-123
subnet zones (DHCP), 384-385
sudo command
  authentication on individual passwords, 128-129
  extending timeout, 126
  individual configurations, 127
  limiting powers with, 123-126
  root password management, 127
superusers (see root users)
supplemental groups, 99, 104
supported filesystems, listing, 246-247
supported printers, finding, 348
suspend mode, 64
suspend-then-hibernate mode, 65
swap files, partition for, 21, 186
switching to root user, 122-123
symbolic notation, 132
  for file permissions, 146-148
  special modes, setting, 148-150
symlinks, 154-157
  for Ctrl-Alt-Delete, 68
  for shutdown commands, 60
syntax checking OpenSSH configuration, 279
system groups
  creating
    with addgroup command, 115-116
    with groupadd command, 110-111
  numbering range, 111
system logs (see logging)
system states, rebooting to, 95-97
system users
  creating
    with adduser command, 114-115

with useradd command, 105
purpose of, 99
systemctl command
disabling firewall, 440
firewalld management, 331
OpenVPN management, 315
services
enabling/disabling, 92-93
killing, 94-95
listing, 86-88
masking/unmasking, 92-93
querying status, 89-90
starting/stopping, 91-92
shutdown with, 60-61
sleep modes, 64-65
symlinks to, 60
targets (runlevels), 95-97
systemd
Ctrl-Alt-Delete configuration, 68-69
determining availability, 82-83
Dnsmasq installation, 375
journalctl, 458-460
journald
configuring, 461-462
logging server creation, 462-464
legacy init systems comparison, 80-81
purpose of, 80
shutdown commands, 60
slow startups, troubleshooting, 98
systemctl command (see systemctl command)
targets, 95-97
timedatectl, 393-396, 404-405
systemd-analyze blame command, 98
systemd-resolved, 368, 375-376
SystemRescue
boot screens, 434-438
bootable device, creating, 432
files, copying over network, 442-445
filesystems/partitions
creating data partition, 451-452
managing, 450
GRUB, repairing, 445-446
preserving changes, 453-454
purpose of, 431
root password, resetting, 439-440
SSH, enabling, 440-441
starting, 432-434
Windows password, resetting, 446-448
SysV init, 79, 80
determining availability, 82-83
runlevels, 95-97

T

tab completion in GRUB, 55
TAP devices, 298
targets (firewalld zones), changing default, 346
targets (systemd), 95-97
tasks, 494
installing, 497
listing, 497
tasksel command, 497
tee command, 226
temperature, monitoring, 465-467
testing
connectivity testing in OpenVPN, 300-302
Dnsmasq from client machine, 380-381
hardware, 479-480
HTTP throughput and latency, 488-490
OpenVPN, 312-314
sites with /etc/hosts file, 371-372
themes for boot screen, 52-53
threads, 84
throughput, testing, 488-490
tilde (~), for /home directory, 139, 167
time servers
chrony configuration, 398-399
ntpd configuration, 403-404
strata, 391-392
time synchronization
chrony
as NTP client, 397-398
as NTP server, 398-399
viewing statistics, 400-401
manual settings with timedatectl, 396
NTP clients, determining, 392-393
ntpd
as NTP client, 401-402
as NTP server, 403-404
overview, 391-392
time zone management, 404-406
timesyncd, configuring, 394-395
time zones, 392, 404-406
timed shutdowns
canceling, 62
with shutdown command, 62
timedatectl, 393-396, 404-405
timesyncd

configuring, 394-395
determining usage, 392-393
purpose of, 391
TLD (top-level domain) name servers, 368
TLS (Transport Layer Security), 298
top command, 94
killing processes, 475-477
viewing processes, 477
top-level domain (TLD) name servers, 368
touch command, 133-136
transparency of colors, 51
Transport Layer Security (TLS), 298
tree command, 133
troubleshooting
Dnsmasq conflicts, 375-376
documentation, purpose of, 455
frozen graphical environments, 478-479
GRUB, repairing from SystemRescue, 445-446
hardware
hard disk monitoring, 470-473
preventing problems, 455-456
temperature monitoring, 465-467
testing, 479-480
logging
journald configuration, 461-462
logging server creation, 462-464
purpose of, 455
severity levels for, 460
viewing log files, 457-460
networks
connectivity tests, 482-484
diagnostic hardware, 481
duplicate IP addresses, 487-488
HTTP throughput and latency testing, 488-490
packet tracing, 490-491
port scanning, 484-486
nonbooting system
grub rescue> prompt, 56-57
grub> prompt, 54-56
patience, importance of, 456-457
printing, 360-361, 366
processes, killing, 94-95
slow startups, 98
sluggish systems, 475-477
trusted zone (firewalld), 338
TUN devices, 298
tune2fs command
configuring journal mode, 257-259
finding block size, 261
freeing reserved space, 262-263
tunneling
with OpenVPN, 297
X sessions over SSH, 288-290
tutorials for Raspberry Pi, 412

U

Ubuntu software management commands, 495-497
(see also Linux)
UEFI (see BIOS/UEFI setup)
ufw (Uncomplicated Firewall), 326, 328
UID (unique identity)
displaying, 101-102
numbering range, 107, 111
umask command, 153-154
umount command, 190, 251
uname command, 240-241
UNetbootin, 7-9
unique identity (see UID)
unit files, parameterized, 315
unmasking services, 92-93
unmounting partitions, 190
upgrading packages with rpm command, 498
USB devices, listing, 235-236
USB stick installation media, creating
data partition for SystemRescue, 451-452
with dd, 13-14
for SystemRescue, 432
with UNetbootin, 7-9
useradd command, 100
default settings, changing, 106-107
human users, creating, 103-104
system users, creating, 105
userdel command, 100, 118-119
usermod command, 100
assigning users to groups, 112
disabling accounts, 118
users
assigning to groups, 112
centralized management, 100
commands for, 100
creating
with adduser command, 113-114
with useradd command, 103-104
deleting
with deluser command, 119

with userdel command, 118-119
disabling accounts, 117-118
files, finding/managing, 120-122
groups (see groups)
nobody, 105, 115
password files, checking integrity, 116-117
privileges, purpose of, 99
root, 99
abilities of, 132
changing sudo authentication, 128-129
changing sudo password timeout, 126
individual sudo configurations, 127
limiting powers with sudo, 123-126
password management, 127
switching to, 122-123
system users
creating with adduser command, 114-115
creating with useradd, 105
purpose of, 99
types of, 99
UID/GID, displaying, 101-102
well-known user directories, customizing, 108-110
UTC (Coordinated Universal Time), 74, 392, 405

V

vcgencmd command, 413
Ventoy, 9
video connections for Raspberry Pi, 417-419
Video directory, customizing, 108-110
viewing (see displaying)
Virtual FAT (vfat), 249
virtual machines, Windows in, 33
visudo command, 123
VPNs (virtual private networks), 297-298
(see also OpenVPN)

W

Wake-on-LAN, remote startups with, 75-77
Wake-on-Wireless LAN (WoWLAN), remote startups with, 77-78
wakeonlan command, 75-77
wakeups (see startup commands)
wall messages, 62
watch command, 466
website performance testing, 488-490
well-known ports, 328
well-known user directories, customizing, 108-110
whois command, 491
wildcard domains in Dnsmasq, 389
Windows
dual-booting, 1, 31-33
OEM product key, recovering, 34
resetting password, 446-448
in virtual machines, 33
Windows Subsystem for Linux 2 (WSL 2), 33
wodim command, 12
work zone (firewalld), 338
WoWLAN (Wake-on-Wireless LAN), remote startups with, 77-78
writing GRUB configuration files, 44-48
WSL 2 (Windows Subsystem for Linux 2), 33

X

X sessions, tunneling over SSH, 288-290
XFS filesystems, 245
creating, 263
resizing, 264-265

Y

yes command, 135

Z

zombie processes, 476-477
zones (firewalld)
changing default, 338
changing default target, 346
creating, 340-341
customizing, 339-340
listing, 332-334
predefined, 337-338
purpose of, 325-326
removing, 341
selecting/setting, 336-338
zypper command, 499-499

About the Author

Carla Schroder is a serial career changer who first laid hands on a PC in the mid-1990s and was instantly hooked. Since then she has worked as a system and network administrator running mixed Linux/Microsoft/Apple networks, tech journalist, and technical writer. Carla has written over 1,000 Linux how-tos for various publications and currently writes and maintains the product manuals for a Linux enterprise software company. She is the author of the *Linux Cookbook* (O'Reilly), *Linux Networking Cookbook* (O'Reilly), and *The Book of Audacity* (No Starch Press). Her fans love her skill at translating nerd to end user, and answering all those "How do I do this?" questions.

Colophon

The animal on the cover of *Linux Cookbook* is a hazel grouse (*Tetrastes bonasia*). Sometimes referred to as the hazel hen, this sedentary bird is one of the smaller members of the grouse species and can be found across much of eastern Europe and northern Asia in dense woodlands.

The patterned plumage of these game birds is more grey in its underparts and more brown on its wings and back. Male hazel grouses have crests on the top of their heads and white-bordered black throats, while female hazel grouses have shorter crests and brown throats. They are ground-feeding birds and eat mainly plant foods as well as insects during the breeding season. Female hazel grouses incubate their eggs and care for their chicks on their own.

The current conservation status of the hazel grouse is "Least Concern." Many of the animals on O'Reilly covers are endangered; all of them are important to the world.

The cover illustration is by Karen Montgomery, based on a black and white engraving from *Meyers Kleines Lexicon*. The cover fonts are Gilroy Semibold and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.

O'REILLY®

There's much more where this came from.

Experience books, videos, live online training courses, and more from O'Reilly and our 200+ partners—all in one place.

Learn more at oreilly.com/online-learning

©2019 O'Reilly Media, Inc. O'Reilly is a registered trademark of O'Reilly Media, Inc. | 175