

Understanding Free and Opensource Operating Systems, The Technology and Philosophy Of - Part I

Jeremy Hajek

07/30/2015

Contents

1	Introduction	10
1.1	Objectives of this book	10
1.1.1	For Instructors	11
1.1.2	For Students	11
1.1.3	Chapter Title Images	12
1.2	Authors and Contributors	12
1.3	Special Thanks	12
2	History of Unix and Linux	13
2.1	Objectives	13
2.2	Outcomes	13
2.3	Zeroith Phase - Where it Began and Why it Matters Now	14
2.3.1	The Kernel	14
2.3.2	Programming Language and Compiler Tools	15
2.3.3	User Interface and User Tools	15
2.3.4	Ken Thompson, Dennis Ritchie, and Bell Labs	15
2.3.5	Assembler and C Language Comparison	17
2.4	First Phase of Unix Maturity – OS Implementation	20
2.5	Second Phase of Unix Maturity – Unix Users, Application Development, and Licensing	20
2.5.1	Richard Stallman and GNU	20
2.5.2	The GNU Manifesto	21
2.5.3	Free Software Foundation and the GPL	21
2.5.4	Free and Opensource State in the 1990s	23
2.5.5	Linus Torvalds and Linux	24
2.5.6	AT&T and BSD Lawsuit	26
2.6	Third Phase of Unix Maturity - Free and Opensource Software Enters the Enterprise	26
2.6.1	Free Software vs. Opensource Software	26
2.6.2	Opensource Software Definition	28
2.6.3	Opensource Licensing	29
2.7	Fourth Phase of Unix Maturity - The Rise of Commercial Linux	29
2.7.1	Modern Linux Distributions	30
2.7.2	The Debian-based Family	30
2.7.3	Red Hat Family	33
2.7.4	Unix and the BSD Family Distros	36
2.7.5	Solaris Based Unix Distros	38
2.7.6	Mobile Based Linux	39
2.8	Fifth Phase of Unix Maturity - Hard Changes to the Nature of Linux	39
2.9	Sixth Phase of Linux Maturity - OpenSource is not a business model	40
2.9.1	GitHub Copilot - Opensource and AI	41
2.10	Chapter Conclusion and Summary	41
2.10.1	Review Questions	42
2.10.2	Podcast Questions	42
2.10.3	Lab	42

3	Installation of Linux	44
3.1	Objectives	44
3.2	Outcomes	44
3.3	Installation of a Linux Distribution	45
3.3.1	Importance of ISOs	45
3.4	Virtual Machines	46
3.4.1	One Ring to Rule Them All... Operating System Rings	46
3.4.2	Virtualization Diagram	46
3.5	Hypervisors	47
3.5.1	TYPE II Hypervisor - Hosted or Desktop Virtualization	47
3.5.2	TYPE I Hypervisor - Bare Metal or Native Virtualization	48
3.6	Installations and ISOs	49
3.6.1	Checksums	49
3.6.2	Planning Your Install	50
3.6.3	Creating Your First Guest Virtual Machine	50
3.6.4	Walk Through the Settings	53
3.6.5	Installing Ubuntu	53
3.6.6	Installing Fedora	54
3.7	Package Managers	56
3.7.1	Traditional Package Managers	57
3.7.2	APT	59
3.7.3	yum & dnf	62
3.8	New Package Managers as App Stores	63
3.8.1	Snaps and snapd	63
3.8.2	Flatpak	64
3.8.3	AppImage	65
3.9	Compiling and Installing source code	65
3.9.1	GNU GCC	65
3.9.2	GNU Make	65
3.9.3	Using Python to Install Python Based Programs	66
3.9.4	Installing VirtualBox Guest Additions 7.x	66
3.9.5	VirtualBox Features	68
3.10	Chapter Conclusions and Review	68
3.10.1	Review Questions	68
3.10.2	Podcast Questions	70
3.10.3	Lab	70
4	Desktop Linux and GUIs	73
4.1	Objectives	73
4.2	Outcomes	73
4.3	From Paper Tape to CLI to GUIs to 4K	73
4.3.1	VT-100	74
4.3.2	Virtual Consoles	74
4.4	Along Comes an X	76
4.4.1	X	76
4.4.2	Project Wayland	78
4.4.3	Ubuntu Mir - Deprecated	79
4.5	Window Managers	80
4.5.1	Compositing	80
4.5.2	Stacking	80
4.5.3	Tiling	80
4.6	Desktop Environments	80
4.6.1	KDE	81
4.6.2	GNOME	81
4.6.3	Xfce	83
4.6.4	LXQT	83
4.6.5	Enlightenment and Lumina	83

4.6.6	Android	83
4.6.7	Who Uses What	83
4.6.8	Gnome 3.3.x Features	84
4.6.9	GNOME 40 Features	86
4.6.10	Installing Windows Managers and Desktop Environments	86
4.7	Conclusion	87
4.7.1	Review Questions	87
4.7.2	Podcast Questions	90
4.7.3	Lab	90
5	The Linux Filesystem, Path, Shell, and File Permissions	94
5.1	Objectives	94
5.2	Outcomes	94
5.2.1	Chapter Conventions	95
5.3	What is a Filesystem	95
5.3.1	POSIX Standard	96
5.3.2	The Linux Standard Base	96
5.3.3	Lennart Poettering’s thoughts on POSIX	97
5.4	Path	99
5.4.1	Absolute Path and Relative Path	99
5.5	The Linux Shell	101
5.5.1	Shell, Terminal, and Commandline	101
5.5.2	Linux Shell Commands	101
5.5.3	Basic Shell Commands	101
5.5.4	Command Nomenclature	103
5.5.5	How to Read Shell Commands and “Speak Linux”	104
5.5.6	History of the Shell	105
5.6	File Permissions and Ownership	106
5.6.1	Owner, Group, Other (World)	107
5.6.2	Reading and Changing File Permissions	107
5.7	Chapter Conclusion and Review	107
5.7.1	Review Questions	108
5.7.2	Podcast Questions	110
5.7.3	Lab	110
6	Shell Meta-Characters, Pipes, Search and Tools	112
6.1	Objectives	112
6.2	Outcomes	112
6.3	Shell Meta-Characters	113
6.4	Standard input, output, and error	115
6.4.1	Standard In	116
6.4.2	Standard Out	116
6.4.3	Standard Error	116
6.5	History and Usage of Pipes	117
6.5.1	Douglas McIlroy	117
6.5.2	Additional Commands Used for Manipulating Standard Out	118
6.5.3	Pipe Usage	119
6.6	Commands for Finding, Locating, and Pattern Matching	120
6.6.1	grep	120
6.6.2	find and locate commands	121
6.7	Hidden files and single dot operator	122
6.8	Chapter Conclusions and Review	122
6.8.1	Review Questions	122
6.8.2	Podcast Questions	125
6.8.3	Lab	125
7	Introduction to Linux Editors, Shell Scripts, and User Profiles	127

7.1	Objectives	127
7.2	Outcomes	127
7.3	History of Unix/Linux Editors	128
7.3.1	Stream Editors	128
7.3.2	Emacs	128
7.3.3	The vi Editor	128
7.3.4	Relationship of vi and vim	129
7.3.5	vi has a Sharp Learning Curve	129
7.3.6	The 3 vi Modes	130
7.4	vi Command Cheat Sheet	130
7.4.1	vi/ex Mode	132
7.4.2	vi/ex Mode Find and Replace Globally	134
7.4.3	Why vi Key Bindings are as They Are	134
7.4.4	Screen Editors	136
7.4.5	GUI Text Editors	137
7.5	Creating Shell Scripts	137
7.5.1	System Path	138
7.5.2	Changing Permissions for Execution	138
7.6	Understanding .bashrc	139
7.7	Chapter Conclusions and Review	140
7.7.1	Review Questions	140
7.7.2	Podcast Questions	141
7.7.3	Lab Chapter 7	142
8	Writing Basic Shell Scripts	144
8.1	Objectives	144
8.2	Outcomes	144
8.3	Basic Shell Scripts - Part II	144
8.3.1	The Bash Shell	144
8.3.2	Shell Script Variables	145
8.3.3	Control Structures	147
8.3.4	IF Statements	147
8.4	Scheduling Shell Scripts With Cron	152
8.4.1	Where to find more	152
8.5	Text Processing with Awk and Sed	153
8.5.1	AWK	153
8.5.2	sed	153
8.6	Chapter Conclusions and Review	154
8.6.1	Review Questions	154
8.6.2	Podcast Questions	156
8.6.3	Lab	157
9	System Administration	159
9.1	Objectives	159
9.2	Outcomes	159
9.3	Sudo and the Root User Paradigm	159
9.3.1	sudo	160
9.4	Logging and Monitoring	163
9.4.1	/var/log/*	163
9.4.2	syslog	164
9.4.3	rsyslog	165
9.4.4	journald and systemd	165
9.4.5	Log rotation	167
9.5	System Monitoring	168
9.5.1	top	168
9.5.2	htop	169
9.5.3	systemd-cgtop	169

9.5.4	atop	169
9.5.5	Additional Monitoring Tools	169
9.6	User Administration	170
9.6.1	useradd	171
9.6.2	userdel and usermod	173
9.6.3	addgroup and groupadd	173
9.6.4	The groups command	173
9.6.5	/etc/passwd	173
9.6.6	chmod	173
9.6.7	chown	174
9.6.8	chgrp	174
9.6.9	The 3 P's of Troubleshooting Linux Problems	174
9.7	Secure Shell	175
9.7.1	RSA keys	175
9.7.2	SFTP	178
9.7.3	SCP	178
9.7.4	Rsync	178
9.7.5	ssh-copy-id	178
9.7.6	SSH config file	178
9.7.7	SSH Service Daemons and Security	179
9.7.8	WireGuard - Linux kernel native VPN	179
9.8	Chapter Conclusions and Review	179
9.8.1	Review Questions	180
9.8.2	Podcast Questions	182
9.8.3	Lab	182
10	Init Services, Daemons, and Processes	185
10.1	Objectives	185
10.2	Outcomes	185
10.3	First Phase of the system boot	186
10.3.1	BIOS	186
10.3.2	UEFI	186
10.4	Second Phase of the system boot	186
10.5	GNU GRUB Boot Loaders	186
10.5.1	GNU GRUB Settings	188
10.6	Third Phase of the system boot	188
10.6.1	SysVinit	188
10.6.2	Working With Services in SysVinit/Upstart	189
10.6.3	Upstart	190
10.6.4	Other SysVinit replacements	191
10.6.5	Systemd and Systemctl	191
10.7	Processes and Services	192
10.7.1	Working With Services in systemctl	192
10.7.2	Major systemd Components	193
10.7.3	Systemd Service Types	194
10.7.4	Killing Processes with systemd	197
10.8	Filesystems /proc	198
10.8.1	Loading Modules	198
10.9	Single User Mode	199
10.10	Chapter Conclusions and Review	199
10.10.1	Review Questions	199
10.10.2	Podcast Questions	201
10.10.3	Lab 10	202
11	Creating, Managing, and Mounting Filesystems	204
11.1	Objectives	204
11.2	Outcomes	204

11.3	Storage Types	205
11.3.1	Mechanical Hard Drives HDDs	205
11.3.2	Solid State Drives SSDs	205
11.3.3	NVMe	206
11.3.4	Virtual Hard Drives	206
11.3.5	Disk Management in VirtualBox	206
11.4	Disk Partitioning and Formatting	213
11.4.1	Drawbacks of disk partitions	218
11.5	Logical Volume Manager	218
11.5.1	Physical Volumes	219
11.5.2	Volume Groups	219
11.5.3	Logical Volumes	219
11.5.4	LVM Snapshots	220
11.6	Filesystems	220
11.6.1	ext/ext2	221
11.6.2	ext3/ext4	221
11.6.3	Giga vs Gibi	222
11.6.4	XFS	222
11.6.5	Next Generation Linux Filesystems	223
11.6.6	Btrfs	223
11.7	Install Btrfs tools	223
11.7.1	Btrfs Creation Commands	223
11.7.2	ZFS	224
11.7.3	ZFS RAID-Z	224
11.7.4	ZFS Snapshots	225
11.7.5	ZFS Send and Receive	226
11.7.6	HFS+, UFS, and APFS	227
11.7.7	F2FS and EROFS	228
11.7.8	DragonFly BSD and Hammer FS	228
11.8	Mounting and Unmounting of disks	228
11.8.1	/etc/fstab	229
11.8.2	systemd Mounting Units	229
11.8.3	Disk related tools	230
11.9	Compression and Archiving tools	231
11.9.1	tar	231
11.9.2	compress	231
11.9.3	gzip	232
11.9.4	bzip2	232
11.9.5	xz	232
11.9.6	zstd	232
11.9.7	tarballs	232
11.10	Chapter Conclusions and Review	233
11.10.1	Review Questions	233
11.10.2	Podcast Questions	235
11.10.3	Lab	235
12	Networking, Webservers, and Intro to Databases	238
12.1	Objectives	238
12.2	Outcomes	239
12.3	Networking	239
12.3.1	IP Addresses	239
12.3.2	MAC Address	239
12.3.3	Comparison of Net-tools and iproute2	240
12.3.4	udev and ethernet naming conventions under systemd	240
12.3.5	Network Configuration Troubles	241
12.3.6	Ubuntu non-netplan Network Manager Config	242
12.3.7	/etc/hosts	244

12.4	Webservers	246
12.4.1	Apache	246
12.4.2	Nginx	248
12.4.3	OpenBSD httpd Process	248
12.4.4	NodeJS	248
12.5	Database and NoSQL	249
12.5.1	MySQL and MariaDB	249
12.5.2	PostgreSQL	251
12.5.3	SQLite	251
12.5.4	MongoDB	251
12.5.5	Network File System - NFS	251
12.5.6	iSCSI	252
12.6	Firewall	252
12.6.1	Firewalld	253
12.6.2	Ubuntu UFW	254
12.7	Chapter Conclusions and Review	254
12.7.1	Review Questions	254
12.7.2	Podcast Questions	256
12.7.3	Lab	256
13	Infrastructure Installation and IT Orchestration	258
13.1	Objectives	258
13.2	Outcomes	258
13.3	Automation and HashiCorp	259
13.4	Vagrant	259
13.4.1	The Problem	259
13.4.2	How Vagrant Benefits You	260
13.4.3	Vagrant Quick Command Tutorial	265
13.5	Packer	265
13.5.1	The Problem Packer Solves	265
13.5.2	Answer Files	272
13.5.3	Putting Vagrant and Packer together	273
13.6	Secrets Management	273
13.6.1	How to Manage Secrets	275
13.6.2	IT Orchestration	276
13.7	Chapter Conclusions and Review	276
13.7.1	Review Questions	276
13.7.2	Podcast Questions	278
13.7.3	Lab	279
14	Final Projects	280
14.1	Objectives	280
14.2	Outcomes	280
14.2.1	Part 1 - Using Ubuntu 18.04	280
14.3	Deliverable	281
14.3.1	Points breakdown	282
15	Appendix A - Standards and Licenses	283
15.1	POSIX Standard	283
15.1.1	Most important things POSIX 7 defines	283
16	Appendix B - Answers for Review Questions	286
16.1	Chapter 01	286
16.2	Chapter 02	286
16.3	Chapter 03 - Review Questions	286
16.4	Chapter 04 - Review Questions	288
16.5	Chapter 05 - Review Questions	289
16.6	Chapter 06 - Review Questions	290

16.7 Chapter 07 - Review Questions	292
16.8 Chapter 08	293
16.9 Chapter 09	294
16.10Chapter 10	296
16.11Chapter 11	297
16.12Chapter 12	298
16.13Chapter 13	299
16.14Chapter 14	300
17 Appendix C - Markdown Code for Podcasts, Labs, and Review Questions	301
17.1 Chapter 02	301
17.1.1 Review Questions	301
17.1.2 Podcast Questions	302
17.1.3 Lab	302
17.2 Chapter 03	303
17.2.1 Review Questions	303
17.2.2 Podcast Questions	305
17.2.3 Lab	305
17.3 Chapter 04	306
17.3.1 Review Questions	306
17.3.2 Podcast Questions	307
17.3.3 Lab	308
17.4 Chapter 5	309
17.4.1 Review Questions	309
17.4.2 Chapter 05 - Podcast Questions	312
17.4.3 Lab	312
17.5 Chapter 06 - Review Questions	313
17.5.1 Podcast Questions	315
17.5.2 Lab - Chapter 06	316
17.6 Chapter 07	317
17.6.1 Review Questions	317
17.6.2 Podcast Questions	318
17.6.3 Lab Chapter 7	318
17.7 Chapter 08	320
17.7.1 Review Questions	320
17.7.2 Podcast Questions	322
17.7.3 Lab	322
17.8 Chapter 09	323
17.8.1 Review Questions	323
17.8.2 Podcast Questions	325
17.8.3 Lab	326
17.9 Chapter 10	328
17.9.1 Review Questions	328
17.9.2 Podcast Questions	330
17.9.3 Lab	330
17.10Chapter 11	331
17.10.1Podcast Questions	334
17.10.2Lab	334
17.11Chapter 12	336
17.11.1Review Questions 12	336
17.11.2Podcast Questions 12	338
17.11.3Lab 12	338
17.12Chapter 13	339
17.12.1Review Questions 13	339
17.12.2Podcast Questions 13	341
17.12.3Lab 13	341
17.13Chapter 14	341

17.13.1 Podcast 14	341
17.13.2 Lab 14	342
17.14 Chapter 15	342
17.14.1 Podcast	342
17.14.2 Lab	342
18 Appendix D - Git Tutorial	343
18.1 Installing Git	343
18.1.1 Windows 10 and 11 - Git Installation via Chocolatey	343
18.1.2 macOS - Git Installation via Homebrew	344
18.2 Git Basics	344
18.2.1 Initial Git Setup	344

Chapter 1

Introduction

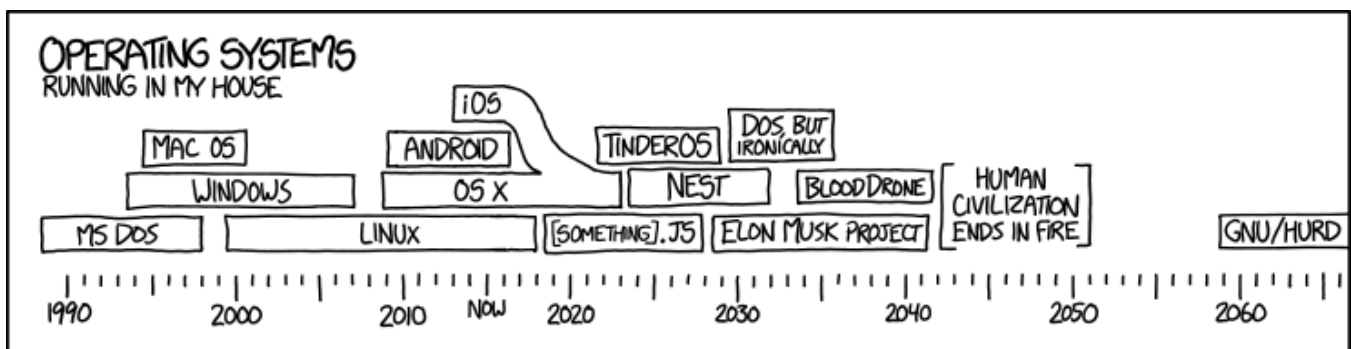


Figure 1.1: One of the survivors, poking around in the ruins with the point of a spear, uncovers a singed photo of Richard Stallman. They stare in silence. “This,” one of them finally says, “This is a man who BELIEVED in something.”

1.1 Objectives of this book

Dennis Ritchie: “I think the Linux phenomenon is quite delightful, because it draws so strongly on the basis that Unix provided. Linux seems to be the among the healthiest of the direct Unix derivatives, though there are also the various BSD systems as well as the more official offerings from the workstation and mainframe manufacturers.” [Interview with Dennis Ritchie](#)

Richard Stallman: “Free software is a matter of liberty, not price. To understand the concept, you should think of free as in free speech, not as in free beer.” [What is free software?](#)

Linus Torvalds: “To kind of explain what Linux is, you have to explain what an operating system is. And the thing about an operating system is that you’re never ever supposed to see it. Because nobody really uses an operating system; people use programs on their computer. And the only mission in life of an operating system is to help those programs run. So an operating system never does anything on its own; it’s only waiting for the programs to ask for certain resources, or ask for a certain file on the disk, or ask to connect to the outside world. And then the operating system steps in and tries to make it easy for people to write programs.” [Linus Torvalds Interview in Revolution OS, documentary, 2001.](#)

Steve Ballmer: “Linux is not in the public domain. Linux is a cancer that attaches itself in an intellectual property sense to everything it touches. That’s the way that the license works.” [1 June 2001 Chicago Sun Times](#)

Scott Guthrie “Today I’m excited to announce our plans to bring SQL Server to Linux as well. This will enable SQL Server to deliver a consistent data platform across Windows Server and Linux, as well as on-premises and cloud. We are bringing the core relational database capabilities to preview today, and are targeting availability in mid-2017.” [March 7, 2016 Scott Guthrie, Executive Vice President, Cloud and Enterprise Group, Microsoft](#)

GitHub Microsoft buys GitHub - June 4, 2018

“Microsoft Corp. on Monday announced it has reached an agreement to acquire [GitHub](#), the world’s leading software development platform where more than 28 million developers learn, share and collaborate to create the future. Together, the two companies will empower developers to achieve more at every stage of the development lifecycle, accelerate enterprise use of GitHub, and bring Microsoft’s developer tools and services to new audiences.”

Android “Android is a Linux distribution according to the Linux Foundation, Google’s open-source chief Chris DiBona, and several journalists. Others, such as Google engineer Patrick Brady, say that Android is not Linux in the traditional Unix-like Linux distribution sense; Android does not include the GNU C Library (it uses Bionic as an alternative C library) and some of other components typically found in Linux distributions.”

When you hear the term “free and opensource software” what comes to mind? The quotes listed above state a range of opinions and outlooks. Do you agree with any of these quotes? Do you disagree? These are the questions to keep in mind as you begin to understand *free and opensource software*.

Everyday people are exposed to free and opensource software without even knowing it. What does this mean? And what are the advantages/disadvantages with proprietary or closed-source software? What are the different development paradigms and what do these things mean in relation to privacy, security, and impact on your daily life. In reading the above quotes it is clear there is more than a technological discussion going on, this is really a philosophy of technology discussion. Operating systems power our computers and smart phones; they allow us to interact with the hardware contained within and to impact society.

In 1969, the first UNIX operating system was created by Bell Labs employees at AT&T. AT&T Bell Labs was spun off as part of Lucent Technologies in 1996. In 2006, Alcatel SA of France, purchased Lucent Technologies and Bell Labs. In 2016, Nokia, with money they received from selling their mobile phone division to Microsoft, absorbed Alcatel SA of France. After one year, the Nokia name was licensed to a HMD Global to produce Android based phones in 2019, which a distribution of Linux, which is descended from UNIX. A 50 year cycle with Unix all the way down.

This text strives to focus on two areas:

- 1) We will describe the philosophy of free and opensource software and its origin and impact upon our computing world.
- 2) This book will cover the basic technical structure and how to use free and opensource operating systems—primarily Linux, but where noted Unix and BSD based systems.

1.1.1 For Instructors

In addition to the text material this book includes:

- Chapter review questions with answers in Appendix B
 - Online quizzes with support for Blackboard importing
- Weekly supplemental podcast questions
- Chapter Labs for reinforcing chapter content
 - Provided in text format
 - For those who use it, provided in Blackboard quiz form – ready for import
- Example programs and source code for exercises

The text of the book is opensource and build instructions are in our GitHub repository for download at <https://github.com/jhajek/Linux-text-book-part-1> Instructors can feel free to fork the repo or submit pull requests. This is to ensure that we have a living document that can be changed, updated, and republished quickly.

1.1.2 For Students

All source code from this book as well as examples are available on our GitHub page: <https://github.com/jhajek/Linux-text-book-part-1>

Part of this book was inspired by a quote from my friend in regards to Linux textbooks, he said, “*Most text books move along well, but then out of nowhere introduce unrelated content.*” At the same time, I had the feeling that most Linux Text books out were actually Unix text books with some updated content relating to Linux.

1.1.3 Chapter Title Images

- Chapter 1 image - http://imgs.xkcd.com/comics/operating_systems.png
- Chapter 2 image - http://imgs.xkcd.com/comics/open_source.png
- Chapter 3 image - <http://imgs.xkcd.com/comics/surgery.png>
- Chapter 4 image - http://imgs.xkcd.com/comics/supported_features.png
- Chapter 5 image - <http://imgs.xkcd.com/comics/seashell.png>
- Chapter 6 image - <http://imgs.xkcd.com/comics/tar.png>
- Chapter 7 image - http://imgs.xkcd.com/comics/real_programmers.png
- Chapter 8 image - <http://imgs.xkcd.com/comics/sandwich.png>
- Chapter 9 image - http://imgs.xkcd.com/comics/security_holes.png
- Chapter 10 image - http://imgs.xkcd.com/comics/debian_main.png
- Chapter 11 image - <http://imgs.xkcd.com/comics/authorization.png>
- Chapter 12 image - http://imgs.xkcd.com/comics/server_problem.png
- Chapter 13 image - <https://xkcd.com/1319/comics/automation.png>
- Chapter 14 image - <http://imgs.xkcd.com/comics/2038.png>
- Chapter 15 image - http://imgs.xkcd.com/comics/tech_loops.png
- Appendix A image - http://imgs.xkcd.com/comics/iso_8601.png
- Appendix B image - http://imgs.xkcd.com/comics/git_commit.png
- Appendix C image - <https://imgs.xkcd.com/comics/authorization.png>
- Appendix D image - <https://imgs.xkcd.com/comics/cautionary.png>

1.2 Authors and Contributors

[Jeremy Hajek](#) - Industry Associate Professor at the Illinois Institute of Technology

Special thanks to those who have submitted pull requests and bug reports:

- [njdwklopper](#)
- [HackingGate](#)
- [ttruty](#)
- [kwillet7](#)
- [paul-arg](#)
- [calennert](#)
- [liamvu2501](#)
- [trevhead0](#)
- [psfales](#)
- [probonopd](#)

1.3 Special Thanks

- [Professor Sam](#) - who taught me that it is pronounced “*ee tee cee F-stab*” not “*etc F S tab*” and how to read error messages.
- Professor Ray Trygstad and Dean Robert Carlson, who gave me my first real IT job and showed me the wonders of Perl.
- Illinois Institute of Technology who has entrusted me with much.
- My wife and kids who have supported me always.
- [Sean Hughes-Durkin](#) who inspired me to write this book based on his quote above.

Chapter 2

History of Unix and Linux

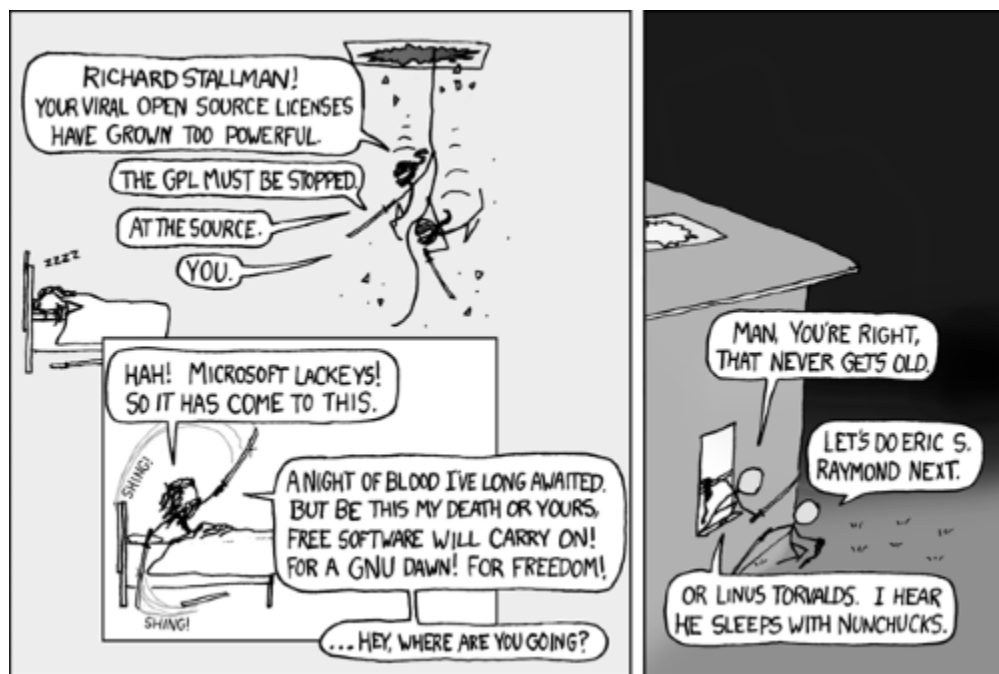


Figure 2.1: At the end of class if your instructor has done their job, you will find this cartoon funny. If you already understand this cartoon you get an “A”.

2.1 Objectives

- Explain the six phases of Unix/Linux maturity and how they encompass each generation’s technological changes
- Explain the contributions of Ken Thompson and Dennis Ritchie to Unix
- Explain the contributions of Richard Stallman to Unix, Linux, GNU, the FSF, and *Free/Libre Software*
- Explain the contributions of Linus Torvalds to the creation of Linux and the Linux Kernel
- Discuss the nature of modern commercial distributions of Linux and Unix operating systems
- Compare and Contrast the principles and key differences of *Free/Libre Software* and *Opensource Software*
- Discuss and explain how Free and Opensource software lead to Cloud Computing and Mobile Compute

2.2 Outcomes

At the completion of this chapter a student will understand and be able to explain the context in which Unix and Linux were created. You will be able to relate key names; *Thompson, Ritchie, Stallman, and Torvalds* to their

respective technological contributions to free and opensource software. You will be able to explain what a Linux distribution is and how Free and Opensource software lead to modern compute paradigms such as Cloud and Mobile. This chapter will also discourse over 6 discreet technology phases since the create of the UNIX operating system.

In this chapter the terms **Linux** and **Unix** are generally interchangeable from a conceptual standpoint. For a large part of this book the conventions are the same - their history is intertwined. Though this book focuses on Linux we would be depriving you of the spectrum of free and opensource software if we left Unix and BSD out. If you are curious, the name is pronounced “*Lin-uks;*” [link to audio pronunciation](#), and Unix; “*Yoo-nix.*”

2.3 Zeroith Phase - Where it Began and Why it Matters Now

One could say that computer programming as we would recognize it started in the 1930’s with Alan Turing. Operating Systems that we would recognize such as Unix didn’t materialize until the early 1970’s. And desktop and server CPU and hardware architecture we are familiar with started in the 1990’s with Intel, and Mobile computing started in late 2000s using ARM based processors.

These modern operating systems such as Windows, MacOS, Ubuntu Linux, Fedora Linux, Android, and others all have a basic architecture in common. At a foundational level all operating systems have three basic components: the kernel, program languages and compiler tools, and user interface and user applications.

2.3.1 The Kernel ¹

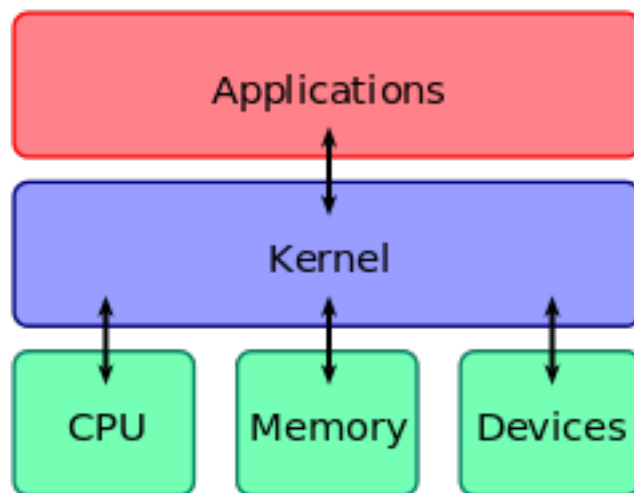


Figure 2.2: *Kernel_Layout.svg*

In the same way all plants require a seed or a kernel to grow, any computer operating system must contain a kernel. This is a small piece of code that forms the core of your operating system. You the user will not interact with the kernel, but devices you use, like a keyboard, mouse, touchscreen, or a Wi-Fi network card will do so when you take any action on the system. How do the devices talk to the kernel? They speak to the kernel via *device drivers*. The figure above describes a two way flow of data; the user interacts with the operating system and the operating system interacts with the kernel. The kernel is the hardware abstraction layer that handles all interfaces from the operating system to the hardware. Without the concept of device drivers and a kernel, each operating system would have to be custom built to communicate with the CPU. The majority of operating system code and drivers are built in the C language.

Take the Windows operating system for instance in which you have just one version, 7, 8, 10, etc. etc. How many of you have an AMD processor? Have an Intel processor? Or an ARM processor? What kind of network card or motherboard brand? You may not even know off the top of your head. There is no need to know because of the Windows kernel abstracting these differences away.

¹By Bobbo (Own work) via [Wikimedia Commons](#)

2.3.2 Programming Language and Compiler Tools

Once you have a kernel to interface with the hardware, you need a programming language and standard library that will let you build tools and libraries to put an operating system together. In Unix and Linux most system tools and commands are built using the C language. These tools such as `gcc` (GNU C Compiler) help you to compile and build other tools and programs.

2.3.3 User Interface and User Tools

All operating systems need a way for a user to interface with the kernel. This is where the *shell* and *user applications* (sometimes called user-land) come into play. The shell is a way for the user to send commands to the operating system—which executes these commands through the kernel and returns output to your screen. Unix originally didn't have a graphical user interface but it always had a shell, which we will cover more about in chapter 5. Even when you click an icon in a GUI, it is actually issuing commands via the shell behind the scenes.

User tools include all tooling or commands (executable binaries) needed to function in an operating system: `copy`, `delete`, `move`, `make directory`, `kill` a process, open a text editor to modify a file, issue a compile command to the C compiler, redirect output from the screen to a file, etc, etc. User applications like web browsers and email clients are seen as user-created tooling that is an amalgamation of many smaller tools to accomplish a larger task.

2.3.4 Ken Thompson, Dennis Ritchie, and Bell Labs

Many people supported and worked on what would become known as the Unix operating system but two names have received most of the credit for the creation, promotion, and use of Unix: **Know these names!**

2.3.4.1 Ken Thompson and Dennis Ritchie

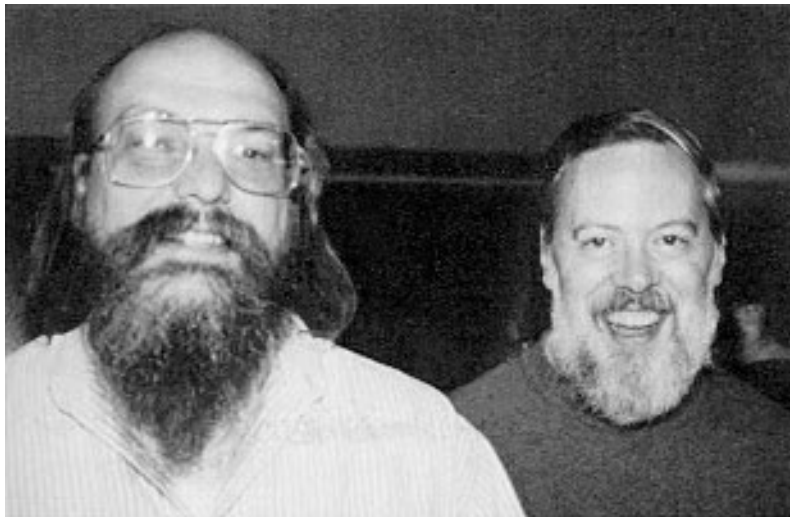


Figure 2.3: *Ken Thompson and Dennis Ritchie*

Without Thompson and Ritchie², there would be no Unix and most likely no Linux today. Until recently both were employed as Distinguished Engineers at Google. Dennis Ritchie passed away in 2011 (same year as Steve Jobs). Ken Thompson is still working and recently helped produce the [Go programming language](#) while at Google.

To begin this story we need to go back to 1968. At the time, the combined might of all the brightest minds of General Electric, MIT, ARPA, and Bell Labs came together to try to build a multi-user operating system called *MULTICS*. Now today those aren't names that come to mind when you think of computer companies. Yet in 1968/1969 General Electric and the government (ARPA) were the large funders and suppliers of computing (The PC market we know of today doesn't come into existence until 1984!).

[Bell Labs](#) at the time was basically the idea shop of the United States—where the best and brightest went to invent everything we take for granted today. Bell Labs was spun off by AT&T and became Lucent Technologies, which

²See [page for author Public domain, via Wikimedia Commons](#)

became Alcatel-Lucent and now is owned by Nokia. One could argue that Google, Netflix, and Facebook have taken its place where the brightest minds go to invent new things in America. No wonder that Dennis Ritchie, Ken Thompson, Brian Kernighan and even James Gosling (creator of the Java programming language) are and were employees at Google.

Like all projects that try to do too much, MULTICS stalled in gridlock between the different companies and the demands of the government. This left one crafty engineer with free time and (for those days) a true rarity - unused computers; PDP-7s to be exact. Ken Thompson had an insiders view of the innovative things MULTICS was trying to accomplish and why the inner workings of the MULTICS project went wrong. Thompson also had a job to do as a Bell Labs researcher. On his own time, he began to use these PDP-7s and program his own multi-user operating system, but with a different twist. It was designed by him, and solved daily work and coding problems he had.

In 1969-70, the “*Unix*” concept had never been attempted. Thompson had no idea that his pet work project was going to become part of a computing revolution. Whereas MULTICS and other computer systems were designed by committees and based on marketable features—due to the nature of the up front financial investment, Unix was simple and easy to build because it solved only a small set of problems—which turned out to be the same problems every engineer had. The overall reason that Unix took hold was that it was designed by engineers to solve other problems that engineers were having—enabling them to get work done.

Unix differences from existing commercial Operating Systems

- Written by Ken Thompson on his spare time
- No company owned it or committee designed it for commercial purposes
- Built by engineers
- Solved problems that engineers had
- Had a consistent design philosophy
- Designed to be portable and work on many hardware vendor platforms

Thompson’s Unix success was also a byproduct of its main design philosophy:

- Everything is a file
 - This means that everything can be read from or written to: all the way from devices to text files
- Unix is portable
 - Tools and OS code can be recompiled for different environments because everything is written in C.
- I/O is redirectable between small executables
 - Small tools that do one and only one thing well
 - Output of one command becomes the input of another command.
 - Complex applications are built by chaining the output of small executables together with *pipes* -> “|”

The best demonstration of these tenets was during a coding challenge issued by [Jon Bently](#) in 1986 to:

Read a file of text, determine the n most frequently used words, and print out a sorted list of those words along with their frequencies.

There were two answers to the problem. [Donald Knuth](#) a preeminent computer scientist, called the “father of analysis algorithms” tackled the problem by originating an ingenious new programming language, lengthy documentation, and code to solve the problem. Comparatively, [Doug McIlroy](#), who was Thompson and Ritchie’s manager, wrote a six line Unix shell script to do the same work Knuth did in his massive work. We will talk more about Doug McIlroy and his contributions to Unix in chapter 6. Here is his answer:

```
tr -cs A-Za-z '\n' |
tr A-Z a-z |
sort |
uniq -c |
sort -rn |
sed ${1}q
```

2.3.4.2 PDP-7

Between 1970 and 1974 Unix grew from a pet project into a real product and one of its crowning achievements—its portability across hardware came to life. Unix was originally written in assembly language for the PDP-7. Its code needed to be as low level as possible because disk storage space was a *HUGE* problem in those days. The code was



Figure 2.4: *PDP-7 restored and running*

good and highly optimized, but the problem with writing in low level assembly is that the code is optimized to only run on a PDP-7 system in this instance. Not on a PDP-11 or a DEC VAX, or an IBM 360, etc, etc.

So what you gain in efficiency you lose in portability. What good would it have been if Unix could only be used on a PDP-7³? It would have stayed a Bell Labs pet project and become an obscure entry on a Wikipedia page today.

While Thompson was building Unix to solve his own workloads, his fellow engineers at Bell Labs got wind of what he was doing and asked to have access to his system. These new people contributed additional functionality to solve their work problems. Enter Dennis Ritchie, who championed Ken Thompson's Unix in Bell Labs. Ritchie was a computer language creator and saw the utility of Thompson's Unix, but realized it was trapped by its use of PDP-7 assembler language. Today we take for granted high level programming languages like C, C++, Python, and Java. In the early 1970's none of these languages existed.

Ritchie's initial work at Bell Labs was on a high level programming language that could would allow a user to write one piece of code and compile it on different computer architectures. In 1970 this was generally not possible and a radical idea. His initial work was on a [language called "B"](#) which was derived from a language called BCPL. B was designed to execute applications and operating system specific tasks but didn't handle numeric data (a feature designed to save precious hard drive space). B was also missing other features you would expect in a modern programming language.

The next logical step was that Thompson and Ritchie went to work extending "B" with all the features they would need to make an operating system fully function and portable. They called this language surprisingly, "C" – the same "C" language you know today. C was different from assembler in that it resembled assembler code syntax but a high enough level of abstraction that the "C" code was an independent language. With the advent of C - Unix was rewritten in this language. With the creation of C compilers for different hardware, Unix could now be built and be recompiled on different architectures, PDP-7, PDP-11, DEC VAX, DEC Alpha, IBM 360, Sun SPARC, eventually Intel and x86, etc, etc.

2.3.5 Assembler and C Language Comparison

[Hello World](#) in x86 Intel Assembly for Linux.

```
section .data
msg db "Hello World!",0x0a
```

³By [en>User:Toresbe](#) via [Wikimedia Commons](#)

```

len equ $-msg

section .text
    global _start

_start:

    mov ebx,0x01
    mov ecx,msg
    mov edx,len
    mov eax,0x04
    int 0x80

    mov ebx,0x00
    mov eax,0x01
    int 0x80

```

Hello World in x86_64 Intel Assembly for Linux

```

hello:
    .string "Hello world!\n"
    .globl _start

_start:
    # write(1, hello, 13)
    mov     $1, %rdi
    mov     $hello, %rsi
    mov     $13, %rdx
    mov     $1, %rax
    syscall

    # _exit(0)
    xor     %rdi, %rdi
    mov     $60, %rax
    syscall

```

C Language equivalent of above code

```

#include <stdio.h>

int main(void)
{
    printf("Hello, world!");
}

```

Since Ritchie had created “C” to solve all the problems Unix had, it became the de facto language of Unix and from that point on pretty much all operating systems were written in “C” after Thompson and Ritchie showed that you could use a high level language to make Unix portable across many platforms. Recently a project to restore the original UNIX assembly code from a tape and documents recovered from 1972 has begun and is [hosted on Google code](#).

2.3.5.1 Brian Kernighan

Thompson didn’t have a name for his project initially, another related figure, Brian Kernighan⁴, can be credited with giving it the name UNIX. This was a play on words–MULTI vs UNI in the name. Kernighan also helped write the original C language textbook along with Dennis Ritchie (Published in 1978, called K&R C which some of you might have used when in school).

⁴By Ben Lowe via Wikimedia Commons

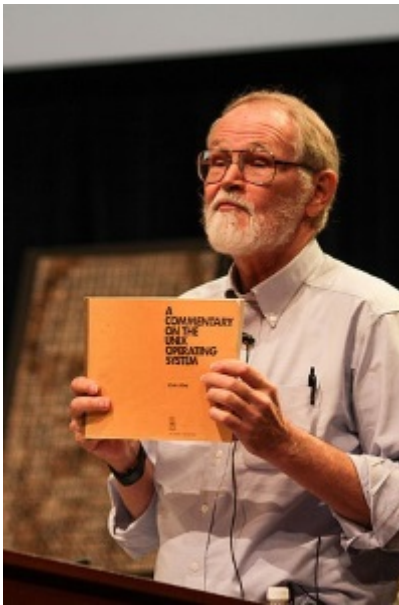


Figure 2.5: *Brian Kernighan in 2012 at Bell Labs*

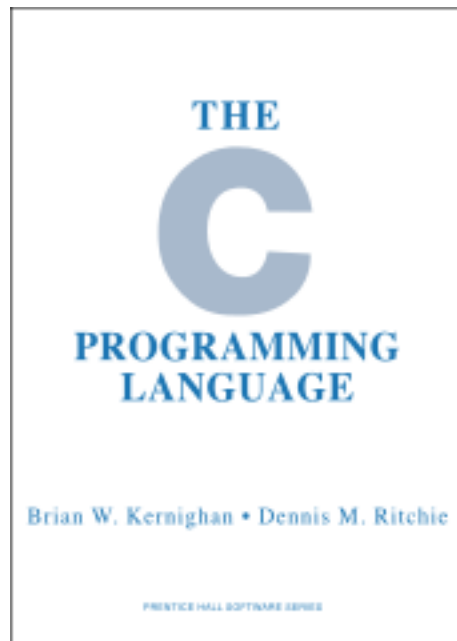


Figure 2.6: *K&R C*

2.4 First Phase of Unix Maturity – OS Implementation

By 1974/75 internal use of Unix at Bell Labs was high amongst engineers. Word was starting to spread about it as Bell Labs engineers moved on and into academia. Other Bell Labs divisions and universities got wind of this and began to request Unix install “tapes” for their own use. *Tapes* meant giant mounted magnetic tape reels that contained all the operating system code. By law AT&T was prohibited from getting into the computer business so they took a hands-off approach to Unix at the time. AT&T left it curiously as Thompson and Ritchie’s pet project. Many universities at this time, wanting to teach computing and operating systems, began to request copies of Unix to teach in their Operating Systems classes. This was attractive to universities because Unix was a fully operational and working system but the main draw was that the source code was freely given away by Ken Thompson. You sent him a letter, paid for shipping, and you got a magnetic tape reel within a week or so. Thompson had no concept of “ownership” and freely shared his project with anyone who was interested. One could say this was Free Software a decade before the term was coined.

In 1975 Ken Thompson took a sabbatical from Bell Labs and went back to his Alma Mater, Berkeley, in California. Installing the Version 6 Unix Release, the students at Berkeley started adding their own features to improve it to solve their own problems. One student in 1978, Bill Joy, added the vi editor and the C Shell (two things still in use today in modern Linux and Unix) and people started referring to the changes made to Unix at Berkeley as BSD Unix (Berkeley Software Distribution Unix).

By 1980, with many copies of Thompson’s Unix now in circulation and nearly a decade of work there started to be fragmentation of the original Unix. At this time the Berkeley Software Distribution of Unix was beginning to vary widely in functionality and quality from commercial AT&T UNIX. Since the code was technically proprietary under AT&T’s ownership there was no way to contribute code back to AT&T. Another interesting problem AT&T had was that by the end of the 70’s all those students who had learned Unix in college went to work in corporations and began to request Unix to be used as their corporate hardware platforms.

Now AT&T had a financial motive to commercialize Unix. In 1982 AT&T released Unix System III, followed by System V in 1983, as a commercial product for sale to companies, while offering a restrictive derivative license for universities. By the dawn of the 1980s the first phase of Unix maturity was complete. The operating system, its code, and its design philosophy were well proven and in wide use across universities and commercial enterprises. In the 1980s the focus of Unix takes a dramatic shift towards application creation.

2.5 Second Phase of Unix Maturity – Unix Users, Application Development, and Licensing

The next phase of Unix history revolves around the important work done by developers to create applications and standards around the Unix operating system to increase productivity and accessibility. There are many people who contributed to this phase but among the names we will discuss there is none more important than Richard Stallman, also known as RMS.

2.5.1 Richard Stallman and GNU



Figure 2.7: *Richard Stallman*

Richard Stallman⁵ was a researcher at MIT in the early 1980’s. He was part of what you would call today a “*hacker culture*” that was constantly researching and developing new tools and applications. They were explorers eager to solve

⁵By Dkoukoul (Own work) , via Wikimedia Commons CC BY-SA 4.0 (<http://creativecommons.org/licenses/by-sa/4.0>)

problems and this lab had a strong culture of sharing software and ideas amongst its students, faculty, and employees. By 1984 a number of small events began to erode this free environment. One such issue was the introduction of account passwords for every computer user. We take passwords for granted these days, but in the early 1980s they weren't standard issue and the implementation of them seemed draconian to users who had always used computers without passwords—freely.

Another event which bothered Richard Stallman was the removal of the capability to modify a network printer's firmware to send an email in case of a paper jam. This was a problem as their networked printer was 2 floors up from the lab and involved quite a walk. The original printer allowed access to their firmware code and Richard Stallman improved the code and added a feature to send emails when a paper jam occurred. The next version of the printer prevented the users from inspecting the firmware and modifying the code—hence locking the lab into a proprietary device. Worse—there was no mechanism for the lab to contribute its improvement to the printer software back to the company.

Stallman disagreed with these concepts as they hindered the ability of the users to inspect the code and improve it, and share those improvements, and suit the software to serve their needs. By 1983 Stallman began to argue that the user's software freedoms were being trampled upon. To make matters worse, in 1984 AT&T began to withhold the source code of Unix and restricted the access of those in the academic world to be able to work with the AT&T source code. Users were now beholden to the closed nature of the products they were using—even if they had paid for the software. Stallman saw this as a **moral** injustice and set about making it his life's work to rectify this issue.

2.5.2 The GNU Manifesto

By late 83/84 Stallman's desire to see free/libre software became a manifesto—an open declaration against proprietary code. This manifesto aimed to reverse engineer all the functionality, capability, and tooling of the then popular Unix operating system **BUT** without the proprietary and restrictive licenses that commercial Unix operating systems entailed, thereby giving the world access to a *free* (as in freedom) operating system. He would call it **GNU** (guh-noo) or *GNU's Not Unix*. Everything done under the GNU project would have the source code *free* for analysis, study, modification, and redistribution. Stallman always intended *free software* to fight perceived injustices relating to software. The term *free software* has nothing to do with cost or money. Through the manifesto, Stallman wanted to let people know his project was Unix-like in functionality and operation and free as in freedom in relation to the source code and redistribution of it.

He felt strongly enough to announce the GNU project publicly in the fall of 1983, and to quit his job working in the MIT lab in 1984 to avoid conflict of interests in pursuit of this work. The [GNU Manifesto](#) was officially published on October 3rd, 1985 and issued a general call to free software arms. Here is the opening paragraph from the manifesto:

2.5.2.1 Why I Must Write GNU

“I consider that the Golden Rule requires that if I like a program I must share it with other people who like it. Software sellers want to divide the users and conquer them, making each user agree not to share with others. I refuse to break solidarity with other users in this way. I cannot in good conscience sign a nondisclosure agreement or a software license agreement. For years I worked within the Artificial Intelligence Lab to resist such tendencies and other inhospitalities, but eventually they had gone too far: I could not remain in an institution where such things are done for me against my will.”

It is easy to write a manifesto but hard to lead a revolution they say. Stallman was one of those rare minds that could do both; a brilliant programmer who had the will and the raw skills and capability to build an entire operating system from scratch and give it away for free and he would get almost all the way there.

“So that I can continue to use computers without dishonor, I have decided to put together a sufficient body of free software so that I will be able to get along without any software that is not free. I have resigned from the AI Lab to deny MIT any legal excuse to prevent me from giving GNU away.”

2.5.3 Free Software Foundation and the GPL

In late 1985 the [FSF](#) – *Free Software Foundation*, was formed to be the holder of all the intellectual property of the GNU project with the motto:

“Our mission is to preserve, protect and promote the freedom to use, study, copy, modify, and redistribute computer software, and to defend the rights of Free Software users.”

With this, they wrote the GPL - GNU Public License, which is a legally enforceable copyleft license. Copyleft is a play on the word copyright, where a copyright restricts the rights of a user. Copyleft, on the other hand enshrines and enforces certain rights to the user of the software. The **4 software freedoms** that the GPL licensed software enshrines are:

- 0) the freedom to use the software for any purpose
- 1) the freedom to change the software to suit your needs
- 2) the freedom to share the software with your friends and neighbors
- 3) the freedom to share the changes you make

In using the term “Free software,” it means software that respects users’ freedom and community. Roughly, it means that the users have the freedom to run, copy, distribute, study, change and improve the software. Thus, “free software” is a matter of liberty, not price. To understand the concept, you should think of “free” as in “free speech,” not as in “free beer”. We sometimes call it “libre software,” borrowing the French or Spanish word for “free” as in freedom, to show we do not mean the software is gratis⁶. You can see a really clear 10 minute presentation of the meaning of Free Software by Richard Stallman at <https://www.youtube.com/watch?v=ZPPikY3uLIQ>.

2.5.3.1 GCC, BASH, and GNU Coretools

In 1984 RMS started his work of creating a free Unix-like operating system. Since Unix was built upon the C programming language, the first thing needed to build a kernel, a shell, or any tooling was a *C compiler*. This project was called GCC (GNU C Compiler) which was a “free” version of the proprietary Unix AT&T cc program (C Compiler).

After GCC was finished, Stallman could begin to build other tools with this *free* C compiler. He needed an extensible text editor to edit files and with which to run GCC to build software. A few years prior, in 1981, James Gosling had created the Gosling Emacs editor; *gmacs*. We will learn more about editors in chapter 7. This name should be familiar because James Gosling would later go on to create the Java programming language while at Sun in 1994. Stallman almost single-handedly rewrote all of Gosling’s code to produce a “free” version of *gmacs* which he called GNU **Emacs**. This program is over 30 years old and still in use and active development to this day.

The next step the GNU project took was to begin reverse engineering all the basic Unix tools and commands needed to accomplish tasks: such as *ls*, *grep*, *awk*, *mv*, *make* and *cp*. Eventually an entire “zoo” of projects was created and are available at [GNU Project website](#). Other GNU tools were added by contributors and volunteers over the years. By 1991 the GNU Bash shell was completed. A shell is the way the user interacts with the operating system to issue command. This was the last major tool that needed completing to have an entire function *free* Unix-like operating system.

The GNU project did remarkably well in quality and quantity of *free* programs considering there were created mostly by volunteers with little funding and no corporate backing. They had pretty much reverse engineered, and in some cases improved, all the components of AT&T Unix by 1991 (8 years of work). The last thing the project was missing was the most critical piece. . . they didn’t have a kernel for their operating system. Turns out that writing a kernel is much harder than it looks.

2.5.3.2 GNU Hurd – The Kernel That Was Not to Be

In their grand ambitions the GNU project was started in 1985 to create a kernel for the soon to come GNU operating system called GNU Hurd. The name GNU Hurd is also another clever recursive hack, as the name GNU has a double meaning. There is a [large goat like animal called a gnu](#), which lives in herds that roam the plains of Africa. The name *Hurd* came from the similarities of a herd of animals and the design of the GNU Kernel would be a “herd” of small micro-processes communicating together, like a herd of gnus (the animal). It seems that GNU developers really love clever hacks. It is something that you have to get used to in opensource as the spirit of bad puns and clever hacks has carried on to this day.

Hurd made some false starts in its initial micro-kernel development phase causing multiple versions to be created and scrapped. What they were trying to do was really innovative but really complicated and difficult to make work reliably. In retrospect Hurd was never finished. By 1998 The GNU project had all but stopped active development and promotion of GNU Hurd as the kernel for its *free* operating system. The GNU project realized that the Linux Kernel had accomplished in four years what GNU had spent 13 years working on.

⁶<https://www.gnu.org/philosophy/free-sw.en.html>

GNU Hurd still exists and is in a usable alpha stage. It is [downloadable today](#) by joining it with the Debian Linux distribution applications—all GPL compliant mind you.

The GNU project instead recommends using the Linux kernel instead and joining the GNU tools with it to form GNU/Linux. In some ways, this realized Stallman’s dream of a *free* operating system and yet his greatest disappointment would be that Linus Torvalds delivered the necessary operating system kernel and not the GNU project. By 1991 a new name comes onto the scene, Linus Torvalds and the Linux kernel come along and make the next leap in the *free* and opensource operating system world by introducing the Linux kernel.

2.5.4 Free and Opensource State in the 1990s

2.5.4.1 Professor Andrew Tanenbaum and Minix

Before we talk about the Linux kernel, we need to talk about the state of other free and opensource operating systems at this time. One in particular is worth mentioning: the Minix operating system. With the closing off of the AT&T Unix source code in 1984, academics and university researchers were left without source code to show as examples in classes.



Figure 2.8: *Professor Andrew S. Tanenbaum*

[Professor Tanenbaum](#)⁷ was teaching at Vrije Universiteit in Amsterdam - and began to write and implement his own *free* Unix-like operating system initially for teaching and research purposes. It was 12,000 lines of C code and system call compatible with commercial Unix. The name [Minix](#) was a combination of “minimal” and “Unix.” Minix 1.0 and 1.5 were released in 1987 and 1991 respectively. Minix 1.0 and 1.5 were freely available to anyone as the source code came in the appendix to a [textbook about operating systems](#) written by Tanenbaum in 1987. Minix was designed to run on older x86 Intel processors (286 and 386 systems) and in version Minix 1.5 a port was included for Sun SPARC processors.

SPARC workstations were common desktop stations used by students and researchers in universities at that time. Any enterprising student could find an old 286 PC or old Sun SPARCstation to run Minix on. The source code for Minix 3 is currently available in a [Git repository](#) and is still being developed and researched. In 1991 many people believed that Minix could have been a viable alternative to commercial Unix and become the still missing GNU Hurd kernel. But the Minix creator, Professor Tanenbaum, was not interested in moving into competition: the code was nowhere near as mature as the Unix code base, which by 1991 had been in existence for almost 20 years! Minix appeared on the radar but was not the missing piece to the GNU puzzle.



Figure 2.9: *Linus Torvalds*

2.5.5 Linus Torvalds and Linux

In 1991, Linus Torvalds was a graduate student at the University of Helsinki in Finland. As a student Torvalds was using Unix for his courses on the university's [Sun SPARCstations](#). He was not pleased with Sun's Unix, called SunOS, but felt it was the best of the commercial Unix available. His real dream was to run his own Unix-like operating system on his own personal PC. He had recently purchased an Intel 386 processor based desktop PC. Linus tried using Minix, but was put off by its minimalist approach and realized it had some good design concepts but was not a complete Unix replacement. In a fashion not unlike Ken Thompson, Torvalds set out in the early part of 1991 to see if he could build his own kernel for his own operating system for his own use and purpose that was Unix-like but wasn't Minix.

This was very impressive feat for a single person. Torvalds himself acknowledged that if GNU Hurd had been ready for production or if at this time AT&T hadn't been suing BSD, he would have re-used their kernel work and not built his own. By August 25th of 1991 the initial release of the Linux kernel was posted online⁸. The quote from the beginning of the chapter was the basis of the initial post to the Minix Usenet Newsgroup (A newsgroup was a bulletin board-like precursor to the actual Internet - like Google Groups in a sense - today you would use Twitter). His initial work was not quite a full-fledged system but really just a small kernel, with a port of the GNU Bash shell, and a port of the GNU C Compiler (GCC). Many things were missing, support for other processors, audio, printers, etc etc but it was like a crack in a damn - something was about to burst forth in the operating system world.

By September of 1991 Linux kernel version 0.01 had been posted to the University of Helsinki FTP servers for public download. The initial kernel included this [release note](#) by Linus:

"Sadly, a kernel by itself gets you nowhere. To get a working system you need a shell, compilers, a library etc. These are separate parts and may be under a stricter (or even looser) copyright. Most of the tools used with Linux are GNU software and are under the GNU copyleft. These tools aren't in the distribution - ask me (or GNU) for more info."

By February of 1992 Linux 0.12 kernel had been released. As more users began to download and use Linux, Linus decided to give the project a proper license for its use. Having seen Richard Stallman speak at the University of Helsinki a few years prior, Linus was inspired and decided to release the Linux Kernel under the GPLv2 license. This was the smartest thing anyone could have done and no one could have foresaw the great economic value the Linux kernel and GNU tools licensed under the GPL has generated for many companies. This combination allowed Google to start and grow to its tremendous size. This combination allowed companies such as Amazon, Yahoo, and Google to get a tremendous start by allowing them to customize the Linux operating system not having to place precious startup capital into Unix licenses.

The reason why Linux is so popular is because of this fledgling kernel that worked enough for people to use, hack on, build upon, and improve now had a governance structure with the GPL that could accept code contributions and be available for commercial work as well. Being GPL the Linux kernel was instantly compatible with all of the entire GNU project's tools base. You instantly had the kernel that GNU was missing and the Linux kernel had all the tools and applications it needed to be usable.

People began downloading and compiling the Linux kernel, adding the GNU tools, and making fully capable Unix-like

⁷By derivative work: Okoura (talk)AndrewTanenbaum.JPG: GerardM (AndrewTanenbaum.JPG) CC BY 2.5, GFDL or CC-BY-SA-3.0, via Wikimedia Commons

⁸By Krd (Own work) CC BY-SA 3.0 or CC BY-SA 4.0, via Wikimedia Commons

operating systems for personal use soon after commercial use. Linus' brilliance comes not from ingenuity but comes from good engineering principals like knowing when not to go down dead-end development trails. Torvalds' work was not perfect but was good enough that others could take it and start to use it and improve upon it. From 1992 to 2001 the Linux kernel grew rapidly in size and features and spawned commercial companies to sell and support Linux Distributions. Stallman's dream of a complete free and opensource operating system was being realized.

This should have been cause for great celebration amongst the Linux and GNU communities. The FSF saw this as a victory for GNU and began calling the system GNU/Linux, assuming that without the GNU tools, the Linux kernel would be useless. The FSF assumed credit in this case. But Linus Torvalds didn't see it that way. He just referred to the system as Linux. He just ignored the FSF's requests and people referred to what should have been GNU/Linux as just Linux, leaving the GNU part out even though all of their tooling is what made Linux possible. In a sense that is Linus' unique personality. On the other hand, Richard Stallman will not conduct any interview unless there is an agreement to only use the term GNU/Linux not Linux. Some would argue that this is Stallman's ego, but he insists he only wants credit where credit is due. This issue is still a matter of contention for the FSF today. Richard Stallman [resigned the presidency of the FSF](#) in September of 2019 and has been replaced by [Professor Geoffrey Knauth](#).

2.5.5.1 Linux Kernel Attributes Compared to Unix OS Attributes

Creation Method

- Linus Torvalds as a graduate student at a university because of his curiosity
- Thompson and Ritchie developed as Bell Labs Engineers to solve computer problems

Release Cycle

- Linux releases a new kernel in short windows and maintains an LTS, Long Term Support Kernel version as well
 - Linux distributions have to plan around this and choose which kernel to use.
- Unix/BSD maintain a complete operating system and release everything together in a 1 to 2 year cycle

License

- The Linux kernel is licensed and protected under the GPLv2 and allows individuals and corporations alike to contribute back to the Linux kernel, but source code must be open and freely available.
- Each BSD project is licensed under a permissive license which allows for derivative works to be made without requiring that the source code be given back to the main project. BSD distros do take outside contributions.
 - Commercial Unix does not take outside contributions

Tooling

- Any Linux based operating system need to make use of a set of coretools – usually the GNU coretools to be able to function
- BSD/Unix may use some GNU coretools but has their own versions internally built in with the distro

2.5.5.2 Linus' Personality

Some people think Linus' personality is a shtick or a comedy act he puts on. But whatever it is he is very straight forward in dealing with people, and will not spare anyone a harsh public rebuking if he thinks they made a sloppy mistake. He justifies this as kernel work is hard and you need to be prepared to take difficult criticism if you are going to survive here. Some consider Linus really mean and even aggressively mean spirited to those with whom he has disagreements. When approached about this, originally Linus states that he only cares about the kernel and no one else matters to him. These links below provide some points and counter points about Linus.

- [Torvalds' right to offend](#)
- [Torvalds doesn't care](#)
- [Linus' response to previous article](#)

On September 16, 2018, Linus Torvalds issued a public apology for his past behavior and temporarily stepped down as Linux Kernel maintainer. You can read the apology letter on the [Kernel mailing list](#). Some say the apology wasn't needed, some say it is too little too late. Others wonder if a man of Linus' age (48) can be reformed. Some see this as a step in the right direction and offer a chance for redemption, some would prefer him never to return. This opens a bigger question of can someone's sins that are on the internet ever be forgiven?

In October 2018, [Linus Torvalds returned from his one month of reflection](#). He realized that his original focus had made the Linux Kernel development area difficult for new people. Torvalds said, “*I need to change some of my behavior, and I want to apologize to the people that my personal behavior hurt and possibly drove away from kernel development.*” In addition a [Contributor Covenant](#) was added to the Linux Project to govern these kinds of interactions in the future.

2.5.6 AT&T and BSD Lawsuit

The nascent Linux project saw a rush of growth and developer contribution from August of 1991 to February of 1992. But where did all these developers come from? At this time we need to go back to Berkeley University and check in on the BSD project, (Berkeley Software Distribution). In the late 80s and up to the early 1990s BSD Unix development had been flourishing at Berkeley. Some would attribute this to great minds and an open environment, some would attribute it to lots of government funding. Either way the product produced began to eclipse the commercial AT&T Unix in features and quality. BSD began to significantly and irreconcilably differ from AT&T Unix.

AT&T seeing this decided in the early 1992 to take the BSD project to court in order to stop BSD from cutting into their commercial business. BSD technically came from AT&T Unix back in 1976, when Ken Thompson took his sabbatical at Berkeley and brought the then pet project AT&T Unix with him. AT&T found that some of the BSD code had not been changed as was claimed and was still original AT&T Unix code, which they claimed was a copyright infringement. In early 1992 AT&T was granted a court ordered development injunction against the BSD project, preventing development work from being done on BSD Unix. This was the perfect time for Linux kernel development to flourish, protected by the GPL, there were no licensing or copyright issues to worry about. BSD developers in droves flocked to the Linux project. By the time the lawsuit was finished in late 1993/1994 it was too late. The results of the court case were that BSD could no longer use the term Unix to describe itself and they had to rewrite a handful of programs to remove AT&T code. AT&T had succeeded in planting the seed for the growth of the entire Linux industry with this action. The Linux rocket had left the launch pad.

After the BSD and AT&T lawsuit was settled the BSD code base split into three and then four main distribution families—each with their own focus but all common enough to share code between them. Also they are free of any contention with commercially licensed Unix and usable for enterprise work. Unlike Linux, BSD lacks a major corporate sponsored distribution, like Ubuntu or Red Hat. All are maintained by volunteer organizations.

- FreeBSD
- DragonFlyBSD, split from FreeBSD
- NetBSD
- OpenBSD, split from NetBSD

2.6 Third Phase of Unix Maturity - Free and Opensource Software Enters the Enterprise

2.6.1 Free Software vs. Opensource Software

By the end of the 90s a curious thing was happening; Microsoft had a near total domination of the desktop PC operating system market and was being investigated by the US Department of Justice for anti-competitive practices. Apple had just brought Steve Jobs back as CEO and was beginning to take steps towards recovering from lost decade. The Internet as the phenomena we recognize today was just beginning to take root in homes across America through dialup services like [AOL](#), [Compuserve](#), and [Prodigy](#). At the same time, the quality and quantity of free and opensource software was increasing. The Internet was being powered by webservers and databases. Opensource tools such as the Apache Webserver, Perl programming Language, and MySQL database became the *killer apps* to use on Linux or BSD.

The term opensource software is pervasive today but until 1998, it didn't exist—they were still using the term *free software*. The term *free software* had existed since 1985, but because of the ambiguity of the English word *free* it became associated with zero-cost free and not freedom free. The term *free* can also give the potential idea of cheap or shoddy work—compared to professional proprietary work and the enterprise would not touch it.

The term *opensource* was not a movement away from the principles of *free software* but a chance to show the enterprise that the opensource development model was sustainable and could produce superior products. To better understand the difference and how it impacts us today, we need to meet some of the key players.

2.6.1.1 Christine Peterson



Figure 2.10: *Christine Peterson*

Christine Peterson⁹ was a technologist who was part of the group that would become the OpenSource Initiative. She is generally credited with coining the term, “*Opensource*,” as a differentiator from “*Free Software*”.

2.6.1.2 Eric S. Raymond

Eric S. Raymond¹⁰ had long been a free software developer and part of the hacker community. Raymond’s most famous contribution to opensource is writing a seminal paper that was later reprinted in book form called, “[The Cathedral and the Bazaar](#)”

His main point was that by business-as-usual practices Linux should have been a massive failure and a poorly implemented experiment. But instead it was an unprecedented success because of the opensource development method. His article examined why this is the case comparing the cathedral like design of Emacs and GCC—open but not publicly available during development versus Linux bazaar style development of everything publicly open and available at all time via the internet.

In conclusion Raymond proposed that opensource code and an opensource design methodology of treating your user as a valued resource was vital to an opensource project’s success. Based on this Raymond and [Bruce Perens](#) founded the [Open Source Initiative \(OSI\)](#) and were part of the group that in 1998 coined the term “opensource”. Their goal was to continue to promote free software but instead of focusing on the moral issue of software freedom, they focused on the design principals of producing superior software. A quote from Raymond puts his opinion bluntly;

As head of the Open Source Initiative, he (Raymond) argued that advocates should focus on the potential for better products. “The”very seductive” moral and ethical rhetoric of Richard Stallman and the Free Software Foundation fails, he said, “not because his principles are wrong, but because that kind of language ... simply does not persuade anybody”. [Eric S. Raymond](#)

The “Cathedral and the Bazaar” was influential in helping the Netscape Corporation opensource their Netscape Web Browser code before the company was sold to AOL, under the name of the Mozilla project. This code gave rise to what would eventually become the Firefox web browser in 2002—thanks to Raymond’s writings.

Richard Stallman and the FSF alleged the OSI was willing to make freedom compromises in order to make larger productivity gains with opensource software and fired back in his article “[Open Source Misses the Point](#)”. The terms do overlap, but Free Software and opensource ultimately have two divergent meanings and *free* software is *not* opensource software.

⁹<http://www.slashroot.in/linux-booting-process-step-step-tutorial-understanding-linux-boot-sequence>

¹⁰By [Eric_S_Raymond_and_company.jpg](#); jerone2derivative work: Bilby ([Eric_S_Raymond_and_company.jpg](#)) CC BY-SA 2.0, via Wikimedia Commons



Figure 2.11: *Eric S. Raymond*

There has been some compromise between the two camps by using the term **FLOSS**, [Free, Libre, and Open Source Software](#), but the FSF does not recommend any licensing that allows a user's freedom to be restricted. One way to conceptualize the difference would be to look at the concept of [DRM software](#), which is basically the copy protection schemes on DVDs and digital music downloads. The OSI group would not be opposed to build opensource DRM software. But the FSF would be opposed to the entire concept of DRM—which is a tool they believe restricts a user's freedom to play their DVD anywhere and in any way.

You can read Raymond's two seminal books on Unix and opensource philosophy online as they are free and opensource licensed:

- [The Art of Unix Usability](#)
- [The Cathedral and the Bazaar](#)

2.6.2 Opensource Software Definition

Open source doesn't just mean access to the source code. According to the [Opensource Initiative](#) written by Bruce Perens, the now 10 rights enshrined in the OSD encompass the 4 software freedoms and extend them with a focus on applications in the business world. The distribution terms of open-source software must comply with the [following criteria](#):

1. Free Redistribution
 - i) The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.
2. Source Code
 - i) The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost, preferably downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the

- output of a preprocessor or translator are not allowed.
3. Derived Works
 - i) The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.
 4. Integrity of The Author's Source Code
 - i) The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.
 5. No Discrimination Against Persons or Groups
 - i) The license must not discriminate against any person or group of persons.
 6. No Discrimination Against Fields of Endeavor
 - i) The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.
 7. Distribution of License
 - i) The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.
 8. License Must Not Be Specific to a Product
 - i) The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.
 9. License Must Not Restrict Other Software
 - i) The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.
 10. License Must Be Technology-Neutral
 - i) No provision of the license may be predicated on any individual technology or style of interface.

2.6.3 Opensource Licensing

The creators of the Opensource initiative (OSI) took a different approach to market free software not as a moral issue, but as a development improvement and business use case. This led to the creation of similar licenses to the GPL, but what was termed as **weak copyleft** or as the Free Software Foundation calls them, *Pushover Copyleft*. These licenses allow for contributions back to the creators as well as the ability to use, share, and inspect the code. They provide one key additional feature in that they let you make a copy and keep the changes—making them proprietary. GitHub has created a site to help you choose a license for your software called, "[Choose a License](#)".

- [MIT](#)
- [APACHE Public License 2.0](#)
- [BSD 3 Clause](#)
- [BSD 2 Clause or FreeBSD license](#)
- [ISC License](#)
- [MPLv2 - Mozilla Public License](#)
- [CDDL](#)
- [OSI Popular licenses](#)

In addition there is the [Creative Commons \(CC\)](#) which covers works that are not considered code. Writings and music are covered by Creative Commons, including all of Wikipedia. Creative Commons has a variety of options you can include that allow for remixing and redistribution or no distribution. There are options that allow for commercial redistribution or prevent it. There are provisions to make sure are credited, and others that are permissive. This way you can choose how your work will be used and contribute to the "[Commons](#)."

2.7 Fourth Phase of Unix Maturity - The Rise of Commercial Linux

As the 1990s came to a close we began to see established companies adopting and using opensource projects in the enterprise, such as MySQL for database and GCC as a C/C++ compiler. Especially we begin to see companies

trying to make commercial distributions of Linux by adding the [GNU core utils](#) and a GUI interface. Of all the Linux companies that started by the turn of the century, Red Hat Linux is one of the few remaining and by far the most successful. How successful? To illustrate this, as of August 10th 2015, Red Hat Linux has a market cap of [~14 billion dollars](#). Most of the Linux distributions started pre-2003 no longer exist.

2.7.1 Modern Linux Distributions

As the new century dawned, Stallman’s dream for the GNU operating system had become a reality. Companies were combining the opensource Linux kernel with the free [GNU core utils](#), and by integrating GUIs such as X11, KDE, and the GNU GNOME project, they were creating what could easily be described as a GNU/Linux based operating system. Each company made their own Linux distribution, also known as a *distro*. As distributions began to proliferate, each distribution began to spawn flavors, derivatives, and different spins as well.

It is curious to see that there were a few small commercial BSD distributions at the same time, but none of them rose to prominence. One might wonder if, without the financial backing of a commercial entity, could a distro ever rise beyond a niche use? BSD distros would argue that mass commercialization was never their primary goal.

As of 2020, we have almost 28 years of Linux Kernel and Linux distribution work. Current major commercial Linux distributions hail from two primary and distinct families: Debian and Red Hat. For the purposes of this book I will focus on these two main distribution families. You can find almost all known Linux distributions at <http://distrowatch.com>.

2.7.2 The Debian-based Family



Figure 2.12: *Ian Murdock - founder of The Debian Project*

“I founded Debian in 1993. Debian was one of the first Linux distributions and also one of the most successful and influential open source projects ever launched. Debian pioneered a number of ideas commonplace today, including employing an open community that allowed (and encouraged!) anyone to contribute (much like how Wikipedia would later operate). And, with its integrated software repositories anyone could contribute to, Debian arguably had the industry’s first (albeit primitive) “App Store”. Today, more than 1,000 people are involved in Debian development, and there are millions of Debian users worldwide.” - <http://ianmurdock.com>

Unfortunately on December 31st, 2015, Ian Murdock took his own brilliant life. Bruce Perens eulogized him in this post: <https://perens.com/blog/2015/12/31/ian-murdock-dead/>. His legacy lives on through the Debian family of distros, which contains 4 major sub-families¹¹.

2.7.2.1 Debian

The Debian distribution (pronounced “*dehb-ian*” officially, but sometimes the stress is put on the first syllable and you will hear “*dee-be-an*”) was founded in 1993 By Ian Murdock and is unique for being one of the major non-commercially

¹¹By Ilya Schurov , Computerra Weekly (originally posted to Flickr as 9722_00_23_14) CC BY-SA 2.0, via Wikimedia Commons



Figure 2.13: *Debian*

backed Linux distro still in existence. The current release is Debian 10. The Debian project has unique characteristics that were designed into the project from the very beginning. Many believe these features are the key to their long term success and usage across the Linux landscape as there are [currently 122 major Debian based distros](#) in existence according to distrowatch.com.

- Initial release schedule was yearly but as Debian project has grown now is two year release schedule
- It is the only major Linux distribution not backed by a corporation.
- Debian is an all volunteer project and organization—project leader is elected on a rotating basis
- Dedicated to protecting software rights and freedoms of users
- First major distribution to come with a [software contract](#)
 - Stating what rights the project will guarantee to the user
- Debian supports free and opensource software by default but will allow for closed source software/drivers to be installed by the user
- Supported at various times 11 different processor types giving it a wide install base
- The Debian project and its history can be found at:
 - [About the Debian Project](#)
 - [History of Debian](#)

2.7.2.2 Ubuntu



Figure 2.14: *Ubuntu Linux*

Ubuntu Linux is downstream distribution¹² based on Debian Linux. Around 2004, [Mark Shuttleworth](#), the founder of Ubuntu, was concerned that Windows had such a dominant position in the PC market. He had been a Debian developer, but felt that the lack of a corporate sponsor in some ways hindered Debian from catching market share from Windows. He set out to make a Debian based distro which he called Ubuntu. Shuttleworth is from South Africa and Ubuntu is a Zulu word meaning “*community*”. Shuttleworth wanted his Linux distro to be people friendly and work really well out of the box—like Windows.

¹²By Macgyu314 (Own work) GFDL or CC BY-SA 3.0, via Wikimedia Commons

In 2004 Red Hat, which had previously owned the desktop Linux market, decided to exit and focus on selling Linux based server operating systems to the enterprise. This left a void that Ubuntu rushed to fill and they did it well. By 2006, Mark Shuttleworth who had started the [Thawte SSL](#) security company, which was bought out by Verisign, took his money and invested 10 million dollars in the Ubuntu Foundation to subsidize the creation and maintenance of Ubuntu Linux¹³.



Figure 2.15: *Mark Shuttleworth*

Ubuntu built on top of rock-solid Debian and extended it by adding closed source drivers and user centered features that were necessary in order to make the best experience. Shuttleworth formed a commercial company called [Canonical](#) that was formed to handle commercial support and the developers who work on Ubuntu.

Ubuntu pioneered the idea of rolling releases - releasing a version of their OS every 6 months. Each distribution is released in late April and late October so there are two distributions per year. Ubuntu also introduced the concept of an *LTS*, Long Term Support - this means that certain releases will have security patches, fixes, and software backported to it for 5 years, allowing you to base an enterprise business off of this product and assured system stability. The LTS releases happen every even year with the April distribution. So Ubuntu 16.04, 18.04, 20.04, 22.04 and so forth (the first number being the year).

2.7.2.3 Kali Linux

[Kali Linux](#) is a Debian-derived Linux distribution designed for digital forensics and penetration testing. It is maintained and funded by [Offensive Security](#)¹⁴.

Cybersecurity has come to the forefront of our daily lives, Kali is a vital tool in that battle. This distro gathers up various tools used for network inspection and intrusion detection.

2.7.2.4 Raspberry Pi OS

[The Raspberry Pi OS](#) is the Debian based operating system for the Raspberry Pi single board computers. The Raspberry Pi project was started in 2011 and Raspberry Pi OS became the official OS as of 2013. The Raspberry Pi project was started to bring the *hacker* ethos back to computing—getting students back to first principles, while being able to run modern Linux operating systems.

2.7.2.5 Other Debian Based Distros

Some of the other notable Debian/Ubuntu based distros are as follows:

- [Xubuntu](#)
 - Ubuntu distro with xfce as the default desktop

¹³By Martin Schmitt (cropped by Mary Gardiner) (<http://www.flickr.com/photos/foobarbaz/141522112/>) CC BY 2.0, via Wikimedia Commons

¹⁴Kali Linux Wikipedia Page

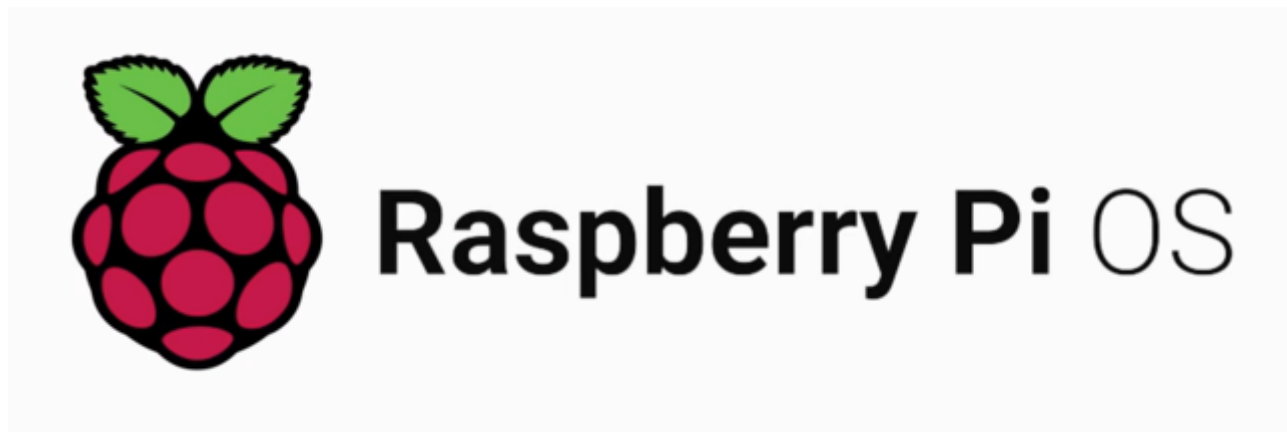


Figure 2.16: *Raspberry Pi OS*

- [Lubuntu](#)
 - Ubuntu distro with lxqt as the default desktop
- [MX Linux](#)
 - Lightweight distro for older laptops and PCs based on Debian Linux
- [Ubuntu Kylin](#)
 - Ubuntu Distro designed for Mandarin Chinese as opposed to English
- [Tails](#)
 - The Amnesic Incognito Live System (Tails) is a Debian-based live CD/USB with the goal of providing complete Internet anonymity for the user
- [Linux Mint](#)
 - Full multimedia support out of the box.
- [Trisquel](#)
 - GNU/Linux-Libre FSF recommended and Richard Stallman uses this one
- [KDE Neon](#)
 - KDE based desktop versioned distro based off of Ubuntu
- [Pure OS](#)
 - This is a Debian derivative, that has been produced with an [FSF](#) based focus on Free Software

2.7.3 Red Hat Family

Red Hat Linux distribution was formed after the Debian project by Marc Ewing and Bob Young. The company had a public IPO on August 11th, 1999. Red Hat being the largest Linux based company was purchased by [IBM in July of 2019](#), for **34 Billion** US dollars. Red Hat source code is currently shared across two main distributions: The Fedora Project and RHEL (Red Hat Enterprise Linux). Currently there are [~25 Fedora based distros](#) or as Fedora calls them “*spins*” – this term is unique to Fedora.

- [About Red Hat](#)
- [Red Hat History](#)

According to Red Hat this is the difference between the various Red Hat operating systems:

Fedora: The upstream project on which future Red Hat Enterprise Linux major releases are based. This is where significant operating system (OS) innovations are introduced.

CentOS Stream: CentOS Stream better connects [ISV](#), [IHV](#) and other ecosystem developers to the OS developers of the Fedora Project—the foundation of the Fedora OS. This shortens the feedback loop and makes it easier for all voices to be heard in the creation of the next Red Hat Enterprise Linux versions.

Red Hat Enterprise Linux: A production-grade operating system that provides a more secure, supported, and flexible foundation for critical workloads and applications.



Figure 2.17: *Fedora Project*

2.7.3.1 Fedora Project

The [Fedora Project](#) was started in 2003 when the Red Hat Desktop Linux product was merged with the company/community based spin off Fedora Core Linux ¹⁵. The [Fedora Project's](#) focus was rapid development and rapid release. They would release two distributions almost yearly, with package and update support only extending back to the previous version cutting off support to viable, but from Red Hat's point of view, outdated software. Remember their focus was rapid iteration of the project to quickly test new technologies. There is a workstation edition, a server edition, a cloud instance, as well as a container based edition [Red Hat CoreOS Edition](#) for use on Kubernetes.

- Fedora 36 was released on 05/06/22
- Fedora 35 was released on 11/02/21
- Fedora 34 was released on 05/20/21
- Fedora 33 was released on 10/27/20
- Fedora 32 was released on 04/21/20

Fedora version less than 35 are no longer supported anymore! Why is the Fedora Project so fast and so merciless on not supporting older versions? This distribution is meant for desktop users and developers who don't mind updating rapidly, called the *bleeding edge*. The Fedora Project is a testing ground for technology that will eventually go into Red Hat's enterprise Linux project, referred to as RHEL.

2.7.3.2 Red Hat Enterprise Linux—RHEL



Figure 2.18: *RHEL*

Red Hat's founder Mark Ewing had been an IBM employee prior to forming Red Hat. He knew something about enterprise software and more importantly enterprise profits. Red Hat began its life as a desktop Linux company. They quickly shifted their focus to compete not with Microsoft and Apple, but to take on the Unix enterprise giants of IBM, HP, Sun and AT&T. There is irony as Red Hat would be purchased by IBM 20 years later. These companies had one thing in common: they were all Unix vendors. Red Hat's vector was to dislodge the established Unix vendors with Red Hat Enterprise Linux (RHEL). They would successfully attack this market with entirely opensource products and running on commodity Intel x86 based processors. With Oracle also sensing a chance to capture market share along with RHEL, it announced it would port its database products to RHEL and this platform became the go to choice for using Oracle as a database. By Oracle doing this they have all but abandoned Solaris and positioned themselves to take on Windows Server and Microsoft SQL Server.

The key to RHEL's success in the enterprise is its long term stability. The RHEL application platform support is 5 years and paid support up to 10 years. An enterprise grade server product cannot be changing every six months like the Fedora project. Red Hat instead takes "snapshots" from the Fedora project and freezes them to produce RHEL versions. For example, RHEL 8.0 was based on Fedora 28. RHEL 9.0 was based on Fedora 34.

How successful is this strategy? By 2012 Red Hat had become the first Linux based company to make a billion dollars in a fiscal year. But this success brought about an additional serious opensource question; RHEL is licensed under the GPLv2 Free Software license, which requires that all source code for your product to be freely and openly available. That means anyone can examine, modify, and redistribute your code for their own product as well. What if someone did that? Wouldn't they be able to ride the coat tails of Red Hat to success? The CentOS project did just that.

¹⁵See page for author [Public domain] via Wikimedia Commons

2.7.3.3 CentOS



Figure 2.19: *CentOS*

By 2010 RHEL was firmly entrenched as a viable enterprise based server platform. Many customers loved the reliability of RHEL, but the two year technology freeze was to long for some people. They wanted to use RHEL but with the opportunity to update libraries and applications much quicker. The CentOS (*Community ENTerprise Operating System*) emerged¹⁶. The goal of this project is to use the freely available GPLv2 code of RHEL and redistribute it with their own custom modifications. Some would argue that CentOS is succeeding based on RHEL's hard work. Until about 2014, Red Hat had a very frosty relationship with the CentOS developers—even taking them to court numerous times over trademarked Red Hat logos that had not been properly removed by CentOS developers. Their developers, like Debian, are entirely volunteer based and not backed by a company (technically since they base their work off of RHEL—they are commercially backed via Red Hat).

Eventually all of Red Hat's copyrighted material was removed and CentOS was then in full compliance with the GPLv2 license. This made Red Hat angry because they felt that CentOS were pirates stealing their work and causing them to lose sales to enterprises that had been using RHEL but had switched to use CentOS. By 2014, Red Hat and CentOS came to terms to work together—with Red Hat offering to sell support contracts to CentOS users. Was CentOS doing anything illegal? Anything immoral? Not according to the GPLv2 and the spirit of free and opensource software.

2.7.3.4 CentOS Stream

Recently Red Hat changed CentOS's purpose and mission. Red Hat had control of CentOS's board of directors for a while and after IBM purchased Red Hat in 2019, Red Hat shifted CentOS to a rolling release called [CentOS Stream](#).

From the [CentOS Blog, December 2020](#):

CentOS Stream will also be the centerpiece of a major shift in collaboration among the CentOS Special Interest Groups (SIGs). This ensures SIGs are developing and testing against what becomes the next version of RHEL. This also provides SIGs a clear single goal, rather than having to build and test for two releases. It gives the CentOS contributor community a great deal of influence in the future of RHEL.

2.7.3.5 Oracle Linux

Oracle saw that many of their customers were paying Red Hat for operating systems licenses, buying support contracts, and running an Oracle database on top of it. Oracle wanted a piece of this pie. Oracle made a fork of RHEL's opensource code as well, adding Oracle product code and services and redistributing it as Oracle Linux.

[Oracle Linux](#) was born in 2007 and is a fully GPL compliant OS. Oracle claims that their “*Unbreakable Enterprise Kernel*” is fully compatible with RHEL, and that Oracle middleware and third-party RHEL-certified applications can be installed and run unchanged. One may ask, isn't this illegal too? Is Oracle breaking the law? Are they stealing RHEL software and reselling it? Is this piracy? Not according to the GPL - they are fully entitled to do this and thus compete with Red Hat selling support contracts on Red Hat's created software—this is the nature of the GPL license.

2.7.3.6 SUSE Linux

- Originally a German company Linux Distro [started in 1994](#)
 - SUSE is an acronym meaning: “*Software und System-Entwicklung*” (software and systems development)
- [openSUSE Linux](#) is a community-driven version of SUSE and released in 2004.
- In 2006 Microsoft and SUSE announce an interoperability agreement (Patent Lawsuit Protection)
- SUSE was responsible for working to port the Linux kernel to 64 bit architecture in 2000
- Major partner for deploying [SAP](#)
- Owns [RancherOS](#) – an Enterprise Kubernetes Management Platform

¹⁶By CentOS Project (<http://wiki.centos.org/ArtWork/Logo/Type>) GPL or Public domain, via Wikimedia Commons

2.7.3.7 Intel Clear Linux

- [Intel Clear Linux](#)
 - Rolling release designed by Intel with auto-updating features built into the OS.
- “*Clear Linux OS is an open source, rolling release Linux distribution optimized for performance and security, from the Cloud to the Edge, designed for customization, and manageability <https://clearlinux.org/>.”*
- Designed to work on recent Intel processors, dropping legacy support for older hardware—in order to optimize for modern cloud and OS Container based workloads like Kubernetes.
- Not focused on desktop Linux but for Cloud-based virtual machines and OS Container images

2.7.3.8 Gentoo Linux

[Gentoo Linux](#)

- Unique Linux distro that you build from scratch based on your unique hardware, to become completely customized, fast, small, and secure.
- Uses the [Portage](#) package system.

2.7.3.9 Alpine Linux

- [Alpine Linux](#)
 - Alpine uses musl as its standard C library not GNU libc, making smaller distros than standard GNU/Linux
 - Designed to be minimal by design - focusing on OS Containers
 - Uses OpenRC init systems
 - Uses [APK](#) Alpine Package Manager

2.7.3.10 Arch Linux



Figure 2.20: *Arch Linux*

Arch Linux has a different philosophy, Arch Linux defines simplicity as without unnecessary additions or modifications. It ships software as released by the original developers (upstream) with minimal distribution-specific (downstream) changes. They also have their own package manager called [Pacman](#).

The current [Steam Deck](#) Steam OS is built using [Manjaro Linux](#) which is based on Arch and different from Ubuntu and Red Hat Oses.

2.7.4 Unix and the BSD Family Distros

While Linux was exploding in the mid 1990s, the AT&T lawsuit against BSD had been settled and work could resume on the BSD forks of Unix. The BSD code splintered into 3 main families or distros that are all descended from the original BSD Unix project. This leaves large pieces of BSD code compatible with each other. BSD is not Linux and technically not Unix but functions in a vary similar manner to Unix. The BSD world is not immune from forks and division.

2.7.4.1 FreeBSD

- Released in November 1994
- Essentially the inheritor of the BSD Unix code base
- Largest BSD implementation used by WhatsApp and Netflix
- No direct commercial backing, instead run by a non-profit foundation
- Legally prohibited from using the term “*Unix*” as the outcome of AT&T lawsuit
- Board of directors are elected and drives development, decisions, and policies



Figure 2.21: *FreeBSD*

2.7.4.2 DragonFly BSD



Figure 2.22: *DragonFly BSD*

- Fork of FreeBSD in April of 2005 by Matthew Dillon.
- Has significantly diverged from the original FreeBSD code base.
- Maintains its own package repositories
- Focused on unique techniques to handle multiprocessing in the FreeBSD kernel
- Introduced a new filesystem called [HAMMER](#) and [HAMMER2](#)

2.7.4.3 Other BSD projects

- [Ghost BSD](#)
- [Hardened BSD](#)

2.7.4.4 OpenBSD



Figure 2.23: *OpenBSD*

- [Theo de Raadt](#) was banned/left from the NetBSD project in 1994.
 - He complained that they were developing too slow and not focusing on security.
- Started a fork of NetBSD at the end of 1995
- OpenSSH, OpenNTPD, OpenSMTPD, LibreSSL, OpenBGPD, and [other projects](#) comes out of this project.
 - [Microsoft recently became the first “gold sponsor” of the project](#)
- Project is focused on radical implementations of security and safe coding practices—leveraging itself as the most secure OS.

2.7.4.5 NetBSD

- Released October of 1994 as another version of the BSD code after the lawsuit.
- Focuses on portability to run this OS on nearly every platform you can think of.



Figure 2.24: *NetBSD*

2.7.4.6 Minix 3

- [Minix 3](#) released October of 2005.
- Since then the OS went from a teaching tool to a product being used commercially.
- Began using NetBSD userland applications for a GUI and package management.
- Intel Management Engine, contained on all modern [Intel CPUs run Minix at Ring -3](#).

2.7.5 Solaris Based Unix Distros

2.7.5.1 Solaris

- Oracle killed commercial Solaris in the middle of the night [September 3rd, 2017](#).
- Commercial Unix distribution created by Sun in the 1980s and bought by Oracle in 2010.
- Sought to merge the best of Sys V and BSD into one standard Unix.
- Ran on proprietary SPARC hardware platform
- Leader in operating system feature development
 - ZFS
 - Dtrace
 - Zones (OS Containers/Jails)
 - Network based installation – Jumpstart
- By 2006 began the process of opensourcing their technology and operating system called OpenSolaris
 - In 2006 Sun had opensourced their Unix-based Solaris OS ¹⁷
 - They hired Ian Murdock (founder of Debian) to oversee this project
 - Project was called OpenSolaris, but was killed when Oracle purchased Sun in 2010
 - [Explanation of how OpenSolaris was killed and closed sourced by Oracle](#).

2.7.5.2 Illumos



Figure 2.25: *Open Indiana*

- After the OpenSolaris project was shut down and Oracle fired most of the Solaris developers, the last version of OpenSolaris was forked into a project called Illumos¹⁸.
- Illumos is not a distro, but a reference implementation in which other OSes are based.
 - [OmniOS](#)
 - [SmartOS](#) released by Joyent¹⁹
 - * Combines the best of the BSD/Solaris technology but runs the best of Linux based desktop applications and software—especially the KVM Virtualization Platform
 - * Recently purchased by Samsung for their OS container technology stack called [Triton](#) and Manta.
 - * Releases all software, even their production cloud as open source.

¹⁷By Sun Microsystems (OpenSolaris) GFDL or CC-BY-SA-3.0, via Wikimedia Commons

¹⁸By openindiana.org [Public domain] via Wikimedia Commons

¹⁹By Joylent (uploaded by [Lamro@enwiki] [Public domain] via Wikimedia Commons

2.7.6 Mobile Based Linux

While Android is a Linux based mobile operating system, it is different from a true Linux operating system. With Android being available and common to develop true Linux based mobile operating systems took a while to get off the ground. Starting in 2018, a movement being lead by [Purism](#), [PinePhone](#), and the previous work done by Ubuntu Touch, there are now Mobile Linux-based distros. These phones are the steps to move a desktop operating system to a mobile arena and all the considerations a phone and tablet have in regards to wireless, battery, and touch input.

- [Ubuntu Touch](#)
- [Mobian - mobile Debian](#)
- [postmarketOS](#)
- [PureOS from Purism](#)

2.8 Fifth Phase of Unix Maturity - Hard Changes to the Nature of Linux



Figure 2.26: *Lennart Poettering*

Not since Linux Torvalds has a single developer been influential in the Linux community. Lennart Poettering is currently a developer for Red Hat²⁰, and had previously developed [PulseAudio](#) and [Avahi](#)—network discovery demon. Lennart is mostly known for *systemd*. Systemd was designed as a modern replacement for SysVinit, the initial program responsible for starting all other system processes in Unix/Linux and systemd was started around 2010 and by 2015 was widely deployed amongst Linux Distros.

Why was this technology be so divisive? Well, the init system is at the root of your entire operating system and defines how the OS operates and interacts with processes. Change is difficult and the init system has been set and well known since 1983. This system is called SysVinit, after the System V AT&T Unix release. Poettering has irked many people by breaking with certain perceived Unix conventions. For him, the Unix philosophy of having small programs do one thing well is a byproduct of bygone era where computers had small amounts of storage and low processing power. He has sentimentality for the work of Thompson and Ritchie, but not if something more efficient can be created. Lennart argues, if Linux wants to be taken serious like MacOS and Windows, these changes are not only necessary but actually closer to the core Unix philosophy than what SysVinit provides.

The other major point of contention is with all the changes systemd makes to the init process. Many other pieces of software need to change as well. Linux has always been about the freedom to choose your software, but systemd has made fundamental choices forcing fundamental changes to the structure of how software interacts with the Linux kernel. For example, the GNOME desktop developers have chosen to integrate their software with systemd directly.

²⁰By Kushal Das (Own work) CC BY-SA 3.0 via Wikimedia Commons

This means that you have no choice but to use `systemd` instead of `SysVinit` if you want to use the GNOME desktop. You are basically locked into using GNOME as a Desktop environment in this case, which some would argue is progress and some would argue as captivity. Note that BSD still uses `SysVinit`; `systemd` is a pure Linux-ism and not compatible with BSD.

You need to give Lennart credit for convincing all major Linux distros to adopt `systemd` as their init system. Lennart convinced his management at Red Hat to take a chance on his technology, and now most of the industry followed suit 6+ years after he first released `systemd`. That is not an easy accomplishment if you think about it.

The first company to adopt `systemd` was Red Hat. Debian was the last holdout and they had a spirited debate, which led to a number of resignations and a split within the community over the issue. Some Debian developers left and went on to form a distro called [Devuan](#)—which is focused on removing all `systemd` and `udev` dependencies from a Debian based Linux distro.

In its defense, `systemd` has many nice and actually new and needed features for Linux. Lennart is updating pieces of Linux that haven't been touched in decades. He even wrote a [21 part defense](#) of `systemd` on his website. We will talk more on the technical aspects of `systemd` in the chapter 11. By using `systemd`, Linux distros make another fundamental choice, to break with Unix based system compatibility. `Systemd` is entirely Linux centric and draws a sharp dividing line between Linux and Unix/BSD based distros. As of July 2022, Lennart left his developer post at Red Hat and [is now employed at Microsoft](#)

2.9 Sixth Phase of Linux Maturity - OpenSource is not a business model

As the story moves into the year of 2019, we begin to see the issue of opensource software and the nature of commercial companies come into conflict. To complicate this, these companies are wide-spread and heavily used across the industry and these companies happen to be Venture Capital funded. These companies, which had relied on opensource software as their business model, have now come into competition directly with large Cloud vendors, mostly Amazon Web Services. These Cloud companies offer cloud-hosted versions of opensource products (legally and correctly) and because of the opensource licenses chosen, don't need to contribute any code or money back to the original product. In some cases the number of developers working on a forked version of a software for the major Cloud providers, vastly out number the original company's developer workforce. What can these companies do? They decided on a licensing vector of attack against the cloud providers.

- MongoDB
 - [Server Side Public License](#)
- Confluent
 - [Community License v1](#)
- Redis
 - [Source Available License](#)
- CockroachDB
 - [Community License Agreement](#)
- Amazon vs Elastic
 - In a bit of irony, after the Elastic company moved some of their products features to an Enterprise Edition and non-opensource offering, Amazon felt the need to make a statement rebuking Elastic for not [keeping Opensource open](#). Amazon neglected to mention that they had forked Elastic's code, made a hosted version on the AWS Cloud and was not contributing any code or money back to Elastic. Mind you this was all legal according to the Apache Public License v2.
- [Grafana changes to AGPL3](#).
 - Some companies have reacted strongly by changing permissive licensing like Apache2 and MIT to the [AGPL3](#), which is within their rights.

Some would argue that this is where the GPL with its strong copyleft would prevent this from happening. Others would argue that this is the fundamental market effect of opensource, and let the two fight it out and the market will determine the winner. The other solution is to create 503c based foundations to steward projects, which means governance and tight corporate control of a project. When Amazon essentially forked MongoDB and created a product called DocumentDB, the GPL would have forced Amazon, by its license, to release the changes made to the MongoDB source code—leveling the playing field. The FSF and GPL people will tell you, the license is not about money or commerce, but freedom.

Others will cite the fact that very few successful software packages are licensed under GPLv2 or v3, with the Linux Kernel being the major exception. Apple, for instance, has removed all GPLv3 based software from its MacOS due to concerns about GPLv3.

One industry solution has been the formation of foundations that have largely avoided the GPL license for fear of the copyleft clause. For example, the [Linux Foundation](#) employs Linus Torvalds to continue work on the Linux Kernel. In addition, the Linux Foundation runs multiple other significant technology foundations—all offspring of Linux and opensource. The [list of projects that are managed](#) and shepherded by the Linux Foundation is pretty astounding.

Other foundations include:

- [Cloud Native Computing Foundation \(CNCF\)](#)

“The Cloud Native Computing Foundation (CNCF) hosts critical components of the global technology infrastructure. CNCF brings together the world’s top developers, end users, and vendors and runs the largest open source developer conferences. CNCF is part of the nonprofit Linux Foundation.”

- [OpenJS Foundation](#): An amalgamation of two other JavaScript Foundations: the JavaScript Foundation [JS Foundation](#) and the [NodeJS Foundation](#).

“The OpenJS Foundation is made up of 32 open source JavaScript projects including Appium, Dojo, jQuery, Node.js, and webpack. Our mission is to support the healthy growth of JavaScript and web technologies by providing a neutral organization to host and sustain projects, as well as collaboratively fund activities that benefit the ecosystem as a whole.”

- The [Continuous Delivery Foundation \(CDF\)](#)

“The Continuous Delivery Foundation (CDF) serves as the vendor-neutral home of many of the fastest-growing projects for continuous delivery, including Jenkins, Jenkins X, Spinnaker, and Tekton. CDF supports DevOps practitioners with an open model, training, industry guidelines, and a portability focus.”

But if you look at the foundations they are “pay-to-play” and controlled by the largest corporations who don’t always have a good track record in relation to respecting free/opensource licensing. Some great technological advances have come about due to foundations but the voice of the individual contributor has been drowned out. The large corporations have found and embraced Opensource Software. The question we need to ask ourselves is, “[What comes next after opensource?](#)”

2.9.1 GitHub Copilot - Opensource and AI

As of 2022 a new wrinkle has appeared in the Opensource discussion. With [GitHub](#) being one of the largest repositories of public opensource code, there came a new opportunity. Could AI scan the existing code base and make coding suggestions as you develop? GitHub is now offering this via their [Copilot](#) paid product.

The question this presents – is this legal? How does the existing opensource licenses: MIT, Apache2, GPLv3, and the MPL impact this service? Can a third party collect opensource code and sell it to you without creating a derivative work? The [FSF](#) has a lengthy commentary on this.

- [On the Nature of AI Code Copilots](#)
- [If Software is My Copilot, Who Programmed My Software?](#)
- [Copilot, Copying, Commons, Community, Culture](#)

2.10 Chapter Conclusion and Summary

In this chapter you were introduced to many names in relation to the history of free and opensource software. We were introduced to the UNIX operating system and how it led directly to the creation of the GNU project, the FSF, Linux, and Free and Opensource software and to the eventual forming of industry foundations. You were introduced to the major Linux distribution families as well as the BSD derivative distros. Finally you were introduced to the six phases of UNIX/Linux maturity and how they relate to business and commercial interests. [Additional Reading on the history of UNIX and BSD can be found here.](#)

2.10.1 Review Questions

Either individually, as a class, or get into groups and watch the documentary movie [Revolution OS - https://www.youtube.com/watch?v=jw8K460vx1c](https://www.youtube.com/watch?v=jw8K460vx1c) made in 2001. The film includes interviews with many of the names that were discussed in this chapter. Watch this movie and answer the questions below or via the online assignment provided for you by the instructor.

1. Based on the movie's tone and rhetoric - why do you think there was an anti-Microsoft tone at the time of the movies making (~2001)? (You may need to research [Microsoft anti-trust case](#)).
2. When Bill Gates wrote his 1976 "Open Letter to Hobbyists", was he justified in his complaint? Why or why not?
3. According to Eric Raymond in the movie, the adoption curve of Linux and the adoption curve the internet do what ~32:20?
4. Why did Bruce Perens help write the Open Source Definition / Debian Social Contract Standard?
5. What were the two commercial Linux companies featured in the movie (Note-one does not exist any longer)?
6. What is Red Hat Linux's stock price today compared to the the price listed in the movie? What is VA Linux's stock price today compared to the movie? (Hint VA Linux was sold and now belongs to another company, find that company's stock price.)
7. According to Eric S. Raymond what was the major application that needed to "flip" for opensource to become a viable enterprise solution?
8. What was the first major commercial company to opensource a key product? What did that product eventually become?
9. How does Richard Stallman react at the end of the movie to the success of the Linux kernel to the exclusion of the GNU toolchain?
10. What is the main difference between "Free Software" and "Open Source Software"?

2.10.2 Podcast Questions

Listen to the Podcast at <https://twit.tv/shows/floss-weekly/episodes/500>

- 7:21 Who is Vicky Brasseur and what does she do?
- 8:45 What has changed the most in regards to the term OpenSource in the last 25+ years?
- 10:00 What is assumed to now be the default development pattern for new software?
- 13:00 According to Randall, when a company publishes its code as OpenSource, what do they gain?
- 15:57 What do companies struggle with when they decide to opensource their codebase and core-products and how do they solve it?
- 19:00 What are some of the OpenSource companies?
- 19:35 What are the majority of OpenSource companies strategies to make money?
- 22:40 What do companies need to figure out about Free and OpenSource?
- 24:25 What is the term "Yak-Shaving" mean?
- 27:25 Who is Vicky's new book targeting?
- 28:03 What is the book about?
- 33:25 What is the mistaken impression about contributing to OpenSource Software?
- 37:40 What do you need to read before contributing to an OpenSource project?
- 43:00 What does Vicky believe is important for a project to have and to enforce for a community?
- 48:01 What is the myth about users of proprietary operating systems (Windows and MacOS)?
- 49:00 Vicky used Linux for 10 years, what drove her away from it?

2.10.3 Lab

2.10.3.1 Activity 1

Most of the popular and functional pieces of software you use everyday involve Free and OpenSource. Choose 2 case studies from <https://highscalability.com> and write a review of the company's architecture based the listed items.

You can find them at <https://highscalability.com/blog/category/example> or look on the High Scalability website on the lower right hand side for the “All Time Favorites” header to find some of the more popular services. It’s best to find a company that you use or support.

Answer these questions (not all of the Answers are in each case study!)

1. What market does that company serve? (What do they do?) And have they always served that market?
2. What Operating System(s) are used?
3. What programming languages/frameworks are used?
4. What storage and what database technologies are used?
5. What Opensource license is the technology stack licensed under?
6. What is the current stock price and what was the IPO of the company? (if traded publicly.)
7. What major obstacle (cost, system performance, QPS, etc, etc) was the company trying to overcome by implementing this technology stack?
8. What can you learn from this article relating to technology and infrastructure?

2.10.3.2 Activity 2

Read these four articles. It’s a commentary on the Opensource license changes, a response from one of the CEOs, and then a reply to the response.

- 1) <http://dtrace.org/blogs/bmc/2018/12/14/open-source-confronts-its-midlife-crisis/>
- 2) <https://medium.com/@jaykreps/a-quick-comment-on-bryan-cantrills-blog-on-licensing-8dccee41d9e6/>
- 3) <http://dtrace.org/blogs/bmc/2018/12/16/a-eula-in-foss-clothing/>
- 4) <https://medium.com/@adamhjk/goodbye-open-core-good-riddance-to-bad-rubbish-ae3355316494>
 - i) <https://sfosc.org/docs/business-models/free-software-product/>

Answer these questions with a few short sentences:

- What is Bryan Cantrill’s initial main point in the first article?
- Do you agree with him, why or why not?
- What is Jay Kreps response/contention in the second article?
- Do you agree with him, why or why not?
- What is the main point of Bryan Cantrill’s rejoinder in the third article?
- Do you agree with him, why or why not?
- What is Adam Jacob’s opinion on running a company with an opensource product?
- What is the solution in your opinion?

2.10.3.3 Footnotes

CC BY-SA 3.0 (<http://creativecommons.org/licenses/by-sa/3.0>) or GFDL (<http://www.gnu.org/copyleft/fdl.html>)

CC SA 1.0 (<http://creativecommons.org/licenses/sa/1.0/>)

CC BY 2.0 (<http://creativecommons.org/licenses/by/2.0>)

Chapter 3

Installation of Linux

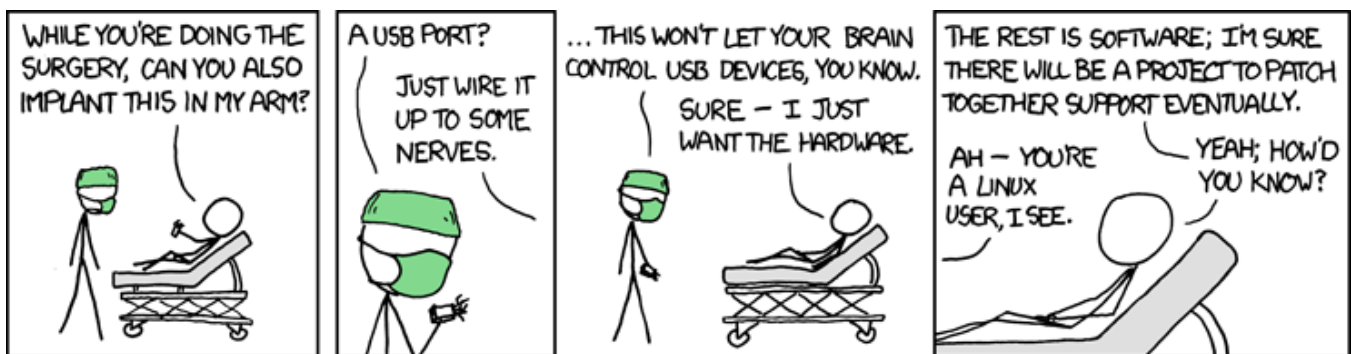


Figure 3.1: *Must be a Linux User...*

3.1 Objectives

- Understand and know how to complete the Linux and BSD Operating System Install Process for the major distributions
- Be able to compare and contrast the two major Linux distribution families' install processes with the BSD install process
- Understand the term *distribution* and *distro*
- Understand the standard installation formats, full ISOs, minimal install ISOs, and net-installer ISOs and the different
- Understand how to use industry standard virtualization platforms for installing distributions
- Learn the features of dnf, yum, and apt to install packages and dependencies in Linux
- Understand how to use GNU tools gcc and make for compiling source code
- Understand how to use the Python language interpreter in order to run Python-based install scripts
- Compiling code from source

3.2 Outcomes

At the end of this chapter you will understand how to complete multiple installations of all the major Linux and BSD platforms. You will be comfortable using the industry standard virtualization platform to enable rapid install of these operating systems. You will also be familiar with and be able to differentiate between processor architectures available. You will be able to install packages and manage dependencies through using standard package managers dnf, yum, and apt. You will also be comfortable compiling software from source using a tar ball and python setup tools.

3.3 Installation of a Linux Distribution

The byproduct of opensource and free software is that the actual source code is freely available. You can download almost all of the major Linux and BSD operating systems at no cost due to the various Free and Opensource licenses. This makes the barrier to entry using a Linux distro very small. As noted previously, in the early days of Linux, installation was not very user friendly. Over the past two decades, however, the various distros have perfected software packaging and installation has since become very simple and extremely user friendly. Later in this chapter, we will install various distributions as part of the lab assignment.

How would one install a Linux distro on a computer? The first thing you need is an *ISO*. An *ISO* (*“eye-so”*) file is actually a standard file type that represents the contents of a CD/DVD-ROM in a single archived file format, like a zip file. Think of ISOs as carrying mechanisms for the content of a disk based installer. The reason the ISO term and format are tied to the Linux Distro installation process is a historical one. During the mid 90s Linux rose to prominence at the same time as CD-ROM technology began to become affordable and became the method of data distribution that replaced floppy and Zip disks amongst PCs. It made sense to create distributions that were almost the exact size of a CD-ROM because it made distributing and copying very easy Linux very easy. In this way Linux spread rapidly. That limitation has been lifted due to the spread of high-speed internet and the deprecation of CD-ROMS in computers.

ISO files can be burned or written to various types of media. CD-ROMs, DVD-ROMs, and more recently USB drives and SD Cards. As of August 2018 you would be hard pressed to find an optical drive on a laptop, 2-in-1s, or even a desktop PC. Though many older and still usable PCs have optical drives. One of the best tools I have found for creating bootable install media on USB drives or [SD Cards](#) is [etcher.io](#).

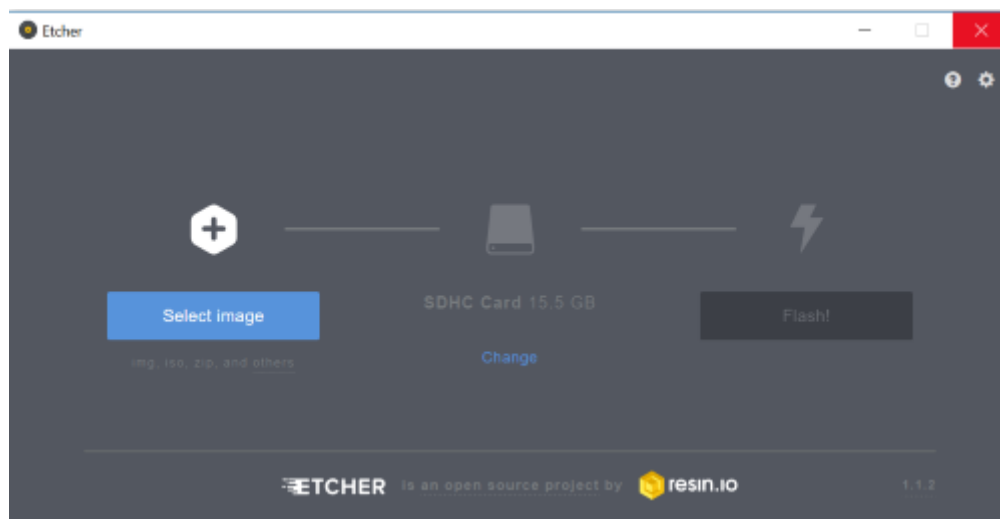


Figure 3.2: *Etcher init screen*

The Fedora Project provides its own media writer which can be used with ISOs and has a download feature built into it. You can download this tool from [the Fedora Project directly](#).

The RaspberryPi Foundation also has their own media writer, [the RaspberryPi Imager](#) which makes dealing with SD Card based Operating Systems very convenient. Their tool also has the auto-download of operating system version feature – which caches the downloads.

One other tool that is of use is Rufus. [Rufus is](#): “A utility that helps format and create bootable USB flash drives, such as USB keys/pendrives, memory sticks, etc.” Rufus is fast and can be used to create Linux images as well as Windows and even DOS boot disks, should the need arise.

3.3.1 Importance of ISOs

This tools takes away the difficulty out of making install media. You may hear the term *“burn”* used in relation to ISOs, all this means is to transfer or write data from one source to its extracted final source.

While you can burn ISO files to media for installation on a desktop or laptop, ISO files also have utility for installing a Linux distribution into a virtualization platform.

3.4 Virtual Machines

Every operating system is made up of multiple components as we mentioned in chapter 2. These components are separated by privilege rings. These rings are for process privilege separation and are built in security for the operating system. With the higher numbered rings being the least privileged. Traditionally user applications are in ring 4 (sometimes called “*userland*” or “*user space*” and the kernel which has the most power is in ring 0. For instance a program a user writes cannot just talk directly to the video card and write to the video memory for the screen. The program needs to go through the OS which in turn goes through the kernel allowing or enforcing commands to be executed. How then does virtualization fit in? Virtualization or called a *hypervisor* is a new ring that inserts itself between the OS and the kernel to intercept commands and is called ring -1.

3.4.1 One Ring to Rule Them All... Operating System Rings

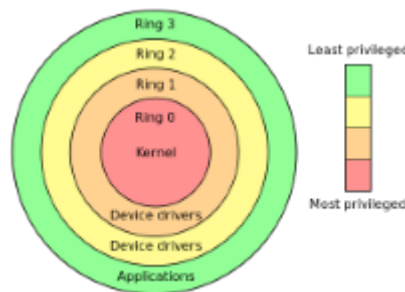


Figure 3.3: *Operating System Rings*

Virtualization works because your CPU¹. To do this, AMD and Intel introduced extensions, *VT-x* and *AMD-V* (called *Pacifica*), to assist virtualization. Both instruction sets add “nine new machine code instructions that only work at ‘Ring -1,’ intended to be used by the hypervisor” ([Andy Dorman - Informationweek](#)). When using virtualization, you are functionally running multiple operating systems at one time. Technically this is not possible as only one operating system can have control of your hardware at a time - so how does a hypervisor and virtualization make this work?

3.4.2 Virtualization Diagram

By having the hypervisor intercepting system calls from the virtualized operating system this allows for multiple operating systems to co-exist on one computer unaware of each other². The way a hypervisor works is not unlike having a professional translator at a business meeting translating between two attendees. The hypervisor essentially creates two classes of operating systems. The “*host*” and potentially multiple “*guests*”. The “*guest*” operating system thinks it has complete control of the hardware - but the virtualization software is only showing the guest system a small portion of all the total RAM, CPU, and disk space available. The hypervisor offers a “*virtualized kernel and drivers*” to the guest operating system. In turn, the hypervisor translates the system commands to the kernel it has received and translates them to the host operating systems commands.

For example if we were running an Ubuntu 14.04.5 Desktop virtualized guest system on a Windows 10 host, the Linux desktop has no way of knowing how to issue a command to use the network card to request a website because Linux knows its own OS and kernel and Windows is a completely different kernel and operating system. The virtualization layer will do the translation for you – allowing the “*host*” system to think that your guest virtualized operating system is nothing more than a native application, and allowing your guest virtualized operating system to think that it owns the entire set of hardware.

The main concept of virtualization you need to know is that your computer (PC, laptop, Mac) has vastly more power than is needed most of the time. The benefit of virtualization is that a hypervisor can act as a translator for multiple operating systems running simultaneously on one system. Thereby maximizing the usage of your resources and

¹Hertzprung at English Wikipedia [GFDL, CC-BY-SA-3.0 or CC BY-SA 2.5-2.0-1.0], via Wikimedia Commons

²Kwesterh

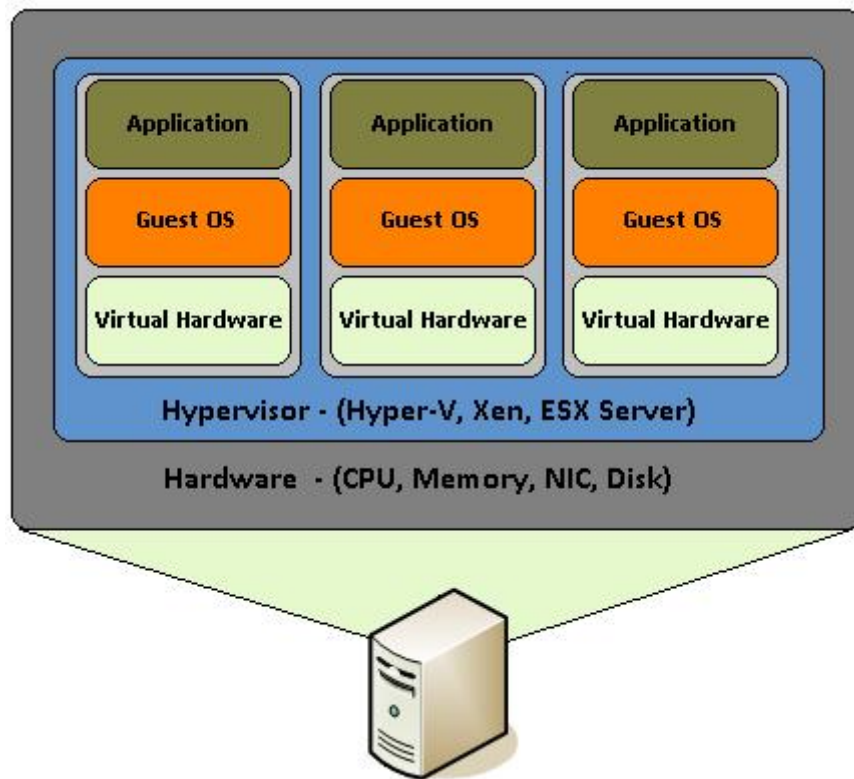


Figure 3.4: *Virtualization Diagram*

preventing you from needing 4 or 5 different physical PCs. See Chapter 13 and 14 for application and development uses of Virtualization.

3.5 Hypervisors

3.5.1 TYPE II Hypervisor - Hosted or Desktop Virtualization

There are basically two types of virtualization. We will be dealing with TYPE II³ exclusively to help us in our mastery of Linux. This type assumes you are using an underlying operating systems such as Windows, Mac, or a Linux distro. In this case the host operating system handles all of the resource management and task scheduling plus you have the added overhead of sharing the hardware with the installed host operating system. In addition you can have multiple hypervisors installed but only one can be active at a time. In general it is best to install only one hypervisor though there are some small edits/hacks that can be done to get [Hyper-V and VirtualBox to run together](#) on Windows. There are four major desktop platforms for virtualization:

3.5.1.1 Microsoft Hyper-V

Hyper-V was originally only a server class product released on Windows Server 2008 R2. Microsoft ported the technology to be able to be used in Windows 8 and 10 Professional and Enterprise editions for free. It comes as an add-on component and is a fully functional implementation of the sever class product. It has the added benefit of being able to work over a wireless connection geared towards desktops. Hyper-V is a good product, if you have an [Xbox One](#), you are using Hyper-V to enable backward compatibility for your games. Hyper-V runs on Windows 8 and 10 and runs Windows, Linux, and BSD virtual machines.

3.5.1.2 Oracle VirtualBox

This product was originally an opensource project that was purchased by Sun and then inherited by Oracle. Though the name is on the project, Oracle has been surprisingly hands off of this project. Because of that it has grown in

³By Scsami (Own work) [CC0], via Wikimedia Commons” <https://commons.wikimedia.org/wiki/File%3AHyperviseur.png>

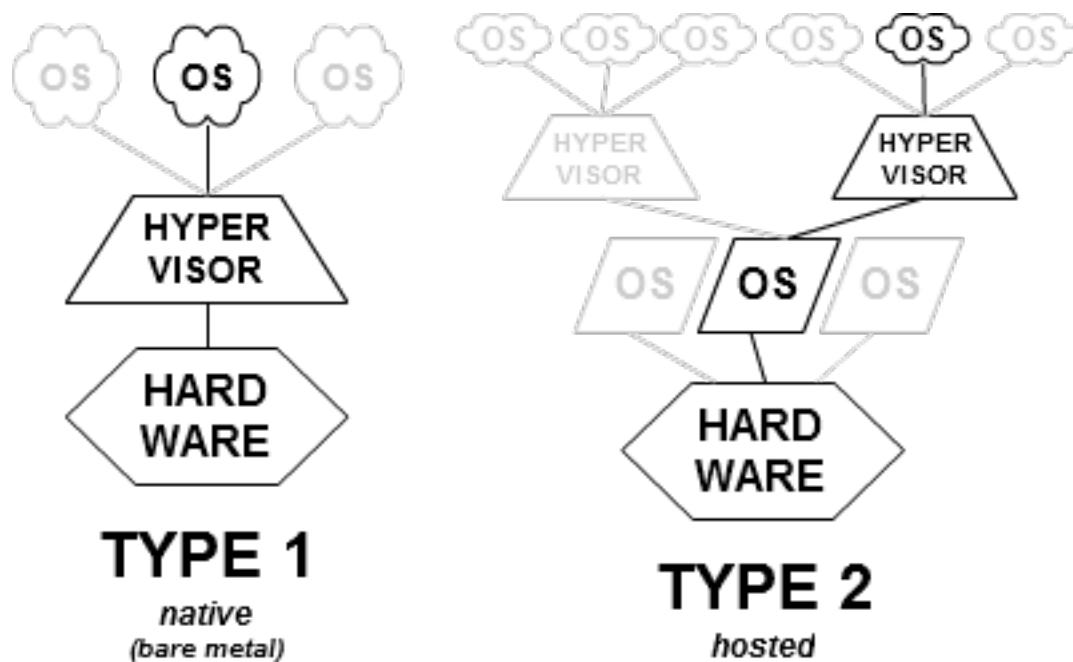


Figure 3.5: *Server and Desktop Virtualization*

usage, features, and utility to become the de facto desktop virtualization tool. It can run on Mac, Windows, and Linux and allows for seamless transfer of virtual machines across platforms

3.5.1.3 VMware Workstation

VMware also released a desktop product that is similar to VirtualBox on Windows and Linux called VMware Workstation. This software predated VirtualBox by nearly 5 years with a separate desktop product available for the Mac called VMware Fusion.

3.5.1.4 Parallels Desktop for Mac

Until 2013 Parallels Desktop was a direct competitor to VMware Workstation on the desktop of Windows and Linux. As of 2013 those products were discontinued in favor of Parallels focusing their desktop product on the Mac.

3.5.1.5 Windows Subsystem for Linux 2

“WSL 2 uses the latest and greatest in virtualization technology to run its Linux kernel inside of a lightweight utility virtual machine (VM). WSL 2 became available in Windows 10 version 2004. It runs on the Home and all of the editions of Windows 10 2004, which is different from the Hyper-V product, which only runs on the Professional edition of Windows 10. WSL 2 ships its own custom optimized Linux Kernel in Windows that virtual machines in WSL 2 will run on. The purpose here is to abstract the underlying hypervisor and provide a simple command line interface to the WSL 2. Note that only one hypervisor can run at a single time so you can use this platform or VirtualBox but not both at the same time, though there is work ongoing to overcome this limitation. <https://devblogs.microsoft.com/commandline/announcing-wsl-2/>.

3.5.2 TYPE I Hypervisor - Bare Metal or Native Virtualization

TYPE I is used in server environments on hardware utilizing multiple core CPUs, multiple terabytes of RAM, and multiple terabytes of hard drive space. A TYPE I hypervisor includes a kernel and tiny purpose driven mini-operating system tuned just for managing and interfacing with virtual machines. The kernel itself is the hypervisor also tuned with all unnecessary features removed. This book will not cover TYPE I hypervisors or commercial implementations of them.

- [Microsoft Hyper-V](#)

- [VMware ESXi](#)
- [KVM](#)
- [XEN](#)
- [bhyve](#)
 - Supported on FreeBSD, macOS, and Illumos

3.6 Installations and ISOs

Now that we covered a bit about what a hypervisor is, let us begin the install process using VirtualBox. The next pages are going to show you in comparison how to install the latest version of [Fedora Workstation](#) and latest [Ubuntu \(LTS\)](#). This will require you to download two ISOs from their respective download sites. For this install process we will assume that you are using VirtualBox version 6.x or later. It might be a good exercise if you have an old laptop or PC laying around to make some installable media (CD-ROM or Flash drive via Etcher) and install one of these distributions directly as the primary operating system. Finding old laptops is easier than you might think. Try asking your relatives (especially around back-to-school and Christmas), a company you work for, or even a school you go to. Laptops/desktops can have a second life and can still be useful to experiment with Linux installations even if the hardware is not the latest or greatest. Another option to consider is dual-booting or triple-booting your system, we will not cover that option here.

If you do try it, back up your data first, you never know what could go wrong. I created a [quad-boot system](#) containing Ubuntu, Fedora, CentOS, and Windows 10. This process is beyond the scope of this book but the link is provided for those interested.

You also need to be aware of the type of architecture you are installing to. In the past we had to determine if the CPU was 32-bit or 64-bit? In modern Linux distros, 32-bit distros are mostly a thing of the past due to all CPUs being 64bit. You can find information about your processor by going to <http://ark.intel.com>. This is Intel's clearing house for all its information about processors and motherboards. They can tell you all you want to know about a processor. All but the most specialized or low end chip these days is 64-bit you should be safe with that type of distro.

The 32-bit distro is most commonly referred to as the x86 or 586, 686 architecture. The 64-bit architecture is usually referred to as x64, but sometimes *x86_64*, and even *AMD_64*, that is not a reference to AMD processors - just a credit in the name as AMD was the first company to implement 64-bit extensions to the 32-bit x86 instruction set—hence the name. There is one other type of architecture called ARM or ARMh7, AARCH, AARCH64. This is the ARM architecture that runs phones, tablets, and small embedded systems such as the Raspberry Pi.

There are ARM based laptops out there such as the [Pinebook Pro](#) and even Apple has moved their laptops to be ARM based, the M1 macs, as of 2021. It has an entirely different instruction set so the software compiled for this architecture is not compatible with the Intel x86-x64 architecture.

Each distro also has a checksum feature provided by the site that issues the download. A checksum is a one way mathematical function that gives you a unique representation of what the content of the ISO should be. That way if you download an ISO from somewhere and the checksum is different then you might be alerted to someone trying to add additional contents or perhaps just a corrupted download. Most distros use the SHA-256 hash, but for legacy purposes you still see md5 hashes.

- [Get Fedora](#)
- [Get Ubuntu](#)
 - [Ubuntu checksum page](#)
 - [Microsoft PowerShell hash checking functions](#)
- [Linux Distro Mirrors](#)
- [Get VirtualBox](#)

3.6.1 Checksums

Here are the commands to execute in Windows in PowerShell:

```
Get-FileHash .\ubuntu-22.04.2-desktop-amd64.iso -Algorithm SHA256 | format-list
```

```
Output: b98dac940a82b110e6265ca78d1320f1f7103861e922aa1a54e4202686e9bbd3
```

Here are the checksum commands and output to be executed if you are running on an already installed version of Linux or Mac OSX from the terminal:

```
sha256sum ./Fedora-Workstation-Live-x86_64-38-1.6.iso
```

Output: 7A444A2E19012023BF0B015AE30135BAFC5FD20F4F333310D42B118745093992

Can you find SHA-256 of the sample PDF located in the book source code, in the folder files -> Chapter-03 -> text-08052020.pdf?

3.6.2 Planning Your Install

Before beginning there are a series of questions you should ask yourself, “What do I need in this distro?”

- Strict Security?
- Ease of use?
- Stable release with long term support?
- Will this be a desktop install or server install? GUI or no GUI?
- What software will you be needing?
 - Serving web pages?
 - Building Android applications?
 - Hacking your neighbor’s Wi-Fi?
 - Developer platform for Coding? Audio and Video?
 - Server edition (no gui)?
 - Building home networks, virtual test beds, Virtual Machine farms?
- What processor do I have 64-bit (Intel or AMD) or ARM (M1 Mac or Raspberry Pi)?
 - How much RAM do I have or need?
- Is this an old PC or laptop I am using—does it lack processor extensions that can aid in rendering media efficiently?
 - [SSE](#)
 - [AVX](#)
- Licensing for business?
 - Does it need to be GPL compliant?
 - Can proprietary programs and codecs be included?

3.6.3 Creating Your First Guest Virtual Machine

Upon completion of a fresh install and launching of VirtualBox you should see this image:

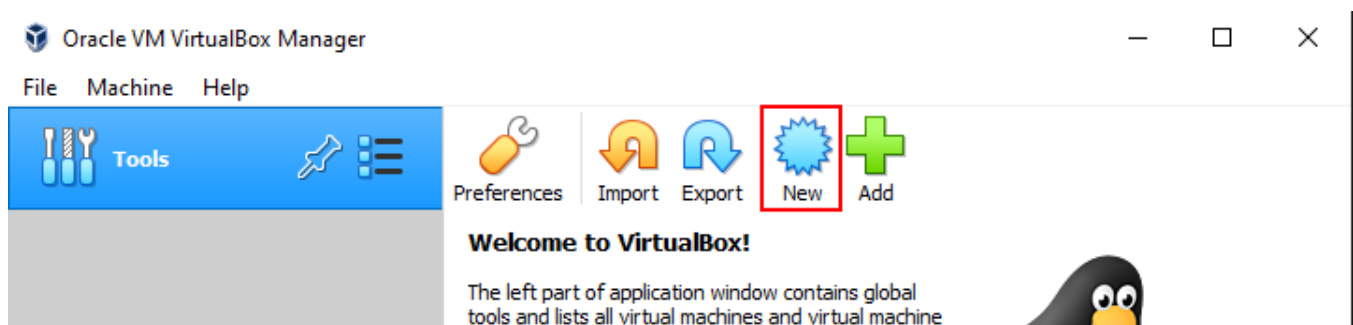


Figure 3.6: *VirtualBox fresh install*

See the [getting started manual](#) for a wide range of information. Unlike some opensource projects this documentation is actually very thorough and useful. VirtualBox has a list of [supported host operating systems](#), which is basically any operating system you can think of from DOS to Haiku to FreeBSD.

Let us walk through the initial installation process. The first step to begin is with the *NEW* button. The next step is where we give our guest virtual machine a name. VirtualBox has long had support that if you type the type of the operating system in the system name - VirtualBox will auto detect the instance type and version. If you see the


sample below I typed “Ubuntu-20-04-Desktop” and “Fedora-34-workstation.” As you type in the title that has the name in it VirtualBox will auto-detect and switch the type and version automatically.

Name and operating system

Please choose a descriptive name and destination folder for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name:

Machine Folder:

Type: 


Version:

Name and operating system

Please choose a descriptive name and destination folder for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name:

Machine Folder:

Type: 

Version:

Figure 3.7: Create New Ubuntu and Fedora Instance Dialogs

What happens if you choose the wrong type or version? Two things: **first**, if you chose the wrong edition of Linux most things will work but the virtualization layer will not be optimized, try to avoid it. You can always go back into the *SETTINGS* menu option and change it after the virtual machine is powered off. **Second**, if you select the wrong version, (32-bit instead of 64-bit) you will receive an error from the BIOS as the operating system loads explaining that it cannot continue.

Next is the amount of memory available - note that this memory is shared with your underlying OS as described with TYPE II hypervisors. Whatever you allocate to this guest VM will be unavailable to the underlying host OS while the guest VM is powered on. Note the slider, each system to install VirtualBox on will have a different slider based on the amount of memory you can allocate. The recommended amount of memory is at least one gigabyte but two gigabytes or more will be better.

Memory size

Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended memory size is **1024 MB**.


 4 MB 16384 MB MB

Figure 3.8: Memory Selection Dialog

The next step is the hard drive creation step. In this step we will choose to create a new hard drive now.

Next is the hard drive file format. There are a few competing standards. If you know you are going to be working in the VirtualBox environment then the default VDI virtual disk type is sufficient.

You can choose to dynamically allocate your hard drive space or statically allocate it. The advantage of dynamically allocating is that not all the space will be assigned right away. The hard drive will grow incrementally as you need space until it hits the maximum you defined.

The final option dialog is where you can choose where to store your virtual machine’s virtual hard drive and the amount of virtual hard drive you want to allocate. Here you have a choice of how much hard drive space you will allocate to the guest VM. This space will be treated as a file by the underlying host OS—allowing for easy migration, export, and even cloning of the guest VM. As a rule of thumb, I generally double the space recommended. Usually the default storage location is fine unless you know you need to store the hard drive on a separate partition or disk.

Now click *finish* and you should be ready to go.

Hard disk

If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.

If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard disk is **10.00 GB**.

- Do not add a virtual hard disk
- Create a virtual hard disk now
- Use an existing virtual hard disk file

Figure 3.9: *Drive Type*

Hard disk file type

Please choose the type of file that you would like to use for the new virtual hard disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.

- VDI (VirtualBox Disk Image)
- VHD (Virtual Hard Disk)
- VMDK (Virtual Machine Disk)


Figure 3.10: *Hard Disk File Type*

- Dynamically allocated
- Fixed size

Figure 3.11: *Disk Type*

File location and size

Please type the name of the new virtual hard disk file into the box below or click on the folder icon to select a different folder to create the file in.



Select the size of the virtual hard disk in megabytes. This size is the limit on the amount of file data that a virtual machine will be able to store on the hard disk.



Figure 3.12: *Hard disk Allocation*

3.6.4 Walk Through the Settings



Figure 3.13: *Virtual Machine Settings*

Before we hit the start button - let us select one of our virtual machines and take a look at the content of the SETTINGS button. Here we will find all the settings possible related to our virtual machine. Though not entirely correct - you could think of this similar to a BIOS settings on a PC - and area where we can configure any underlying hardware. The most common options you will deal with are described as follows:

General Setting that handle meta-data about the virtual machine

System Settings to change amount of RAM and processors - enabling or disabling chipset drivers and CPU features

Display Settings to change how the display settings work

Storage Settings related to disk drives.

Network Settings related to changing or adding new network interfaces or network types.

3.6.5 Installing Ubuntu

Hitting the *START* button on your virtual machine for Ubuntu 20.04 Desktop will bring you to a screen that asks you to select install media (or ISO file):

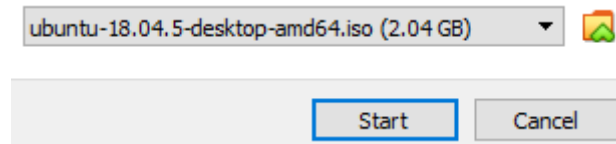


Figure 3.14: *Select ISO install media*

WELCOME screen. Here you will be presented with the option to choose your install language. As well as presented with two options: *TRY UBUNTU* or *INSTALL UBUNTU*. The option *TRY UBUNTU* will load the actual Ubuntu operating system but load it into RAM and not install it on your hard drive. This is helpful because it gives you the option to use Ubuntu fully without permanently installing it. Note that all data is stored in memory so nothing will survive a reboot - but this may be a good tool for doing online banking with. The Live option also has the option to go into an install mode via a desktop icon.



Figure 3.15: *Live or Install Mode*

For installation type you will be presented with default options such as *ERASE DISK AND INSTALL UBUNTU*. In addition there are options for enabling full disk encryption for securing your install data. The third option listed is to install using the LVM method of partitioning, which will be covered in chapter 12. The fourth option is for a custom

partitioning, which is helpful in cases when you are installing multiple operating systems and create a multi-boot system.

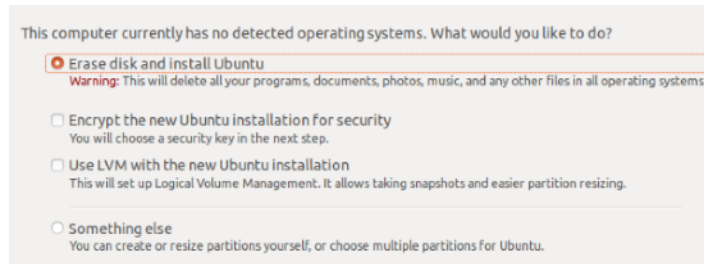


Figure 3.16: *Confirm Partitions*

At the completion of this dialog box you will be asked to confirm the automatically generated partitions created by the system. A Linux system needs a minimum of 2 partitions to function but 3 are recommended. Those partitions are / (pronounced root), /boot (where all the files needed to start the OS are located, and swap (which is an on disk based RAM supplement partition).



Figure 3.17: *Installation Type*

The next parts of the installation contain pretty straight-forward parts: time zone selection, keyboard layout, and account creation and password setup. Note that Ubuntu doesn't create any root user by default, which ever user you create first is automatically placed into the **sudo** group, which is a super user. For a discussion of password strength and strategies see [this cartoon](#). You can also see the install details by clicking the small white triangle to reveal the verbose output of the process

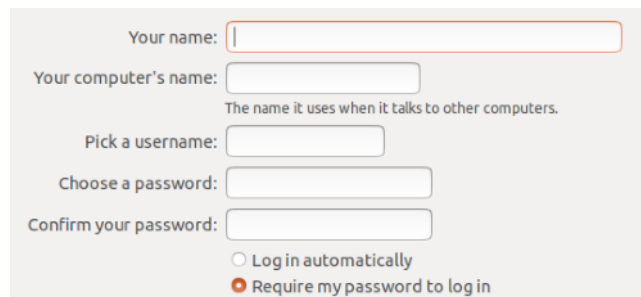


Figure 3.18: *Install Details*

3.6.6 Installing Fedora

Similarly on Fedora 32 you will be presented with the option to *Start Fedora Live* or go into troubleshooting mode. You will be presented with an install screen similar above: *TRY FEDORA* or *INSTALL TO HARD DRIVE*. Fedora 32 will initially present you with a language screen option. After choosing your default language the next step is the *installation summary*.

You will note 3 categories: keyboard, time and date, and installation destination. The first three options should all be

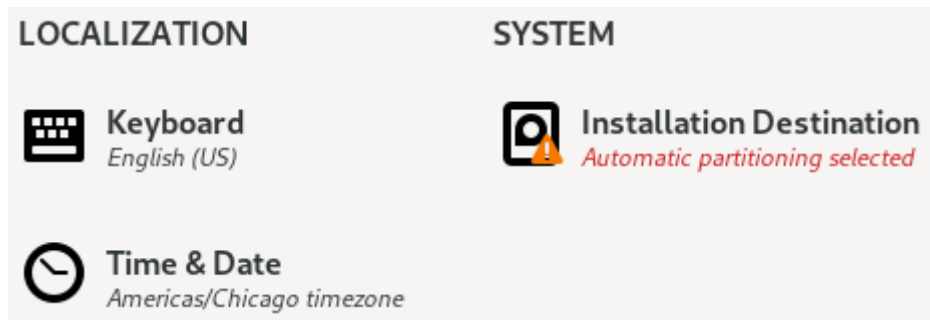


Figure 3.19: *Installation Summary*

filled out by default, the last option installation destination will have an orange notification icon next to it. This means we need to double click on this section and enter a sub-menu before we can continue.

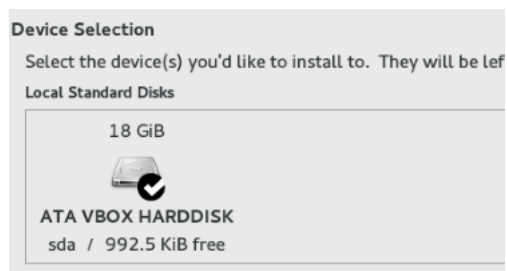


Figure 3.20: *Device Selection*

This warning icon is a forcing mechanism to make you review these settings. You will see visual icons of all the hard disks available for installation. The one with a white checkmark is the disk where the “/” (root) partition will be installed. If the default selections are satisfactory then you can click **DONE** button at the top to the screen and continue. Otherwise at the bottom of the screen are the detailed installation options, which include encryption, LVM, and external installation drives such as over iSCSI or NFS.

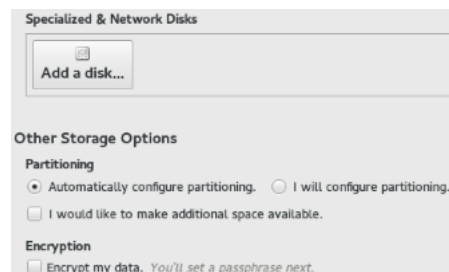


Figure 3.21: *Specialized Details*

Now you can click “*begin installation*”.

The installation will finish without prompting you for creating a user. This will be done on the subsequent reboot as part of the setup process. You will be prompted with optional dialogs, and then a user creation dialog where you will create a user account that will be given admin privileges and/or the ability to log into a corporate account that is managed by a central LDAP or Active Directory (single sign-on).

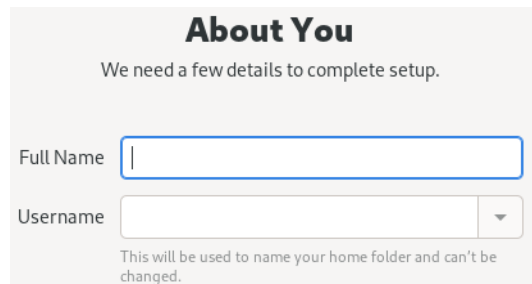


Figure 3.22: *Create User With Sudo*

3.7 Package Managers

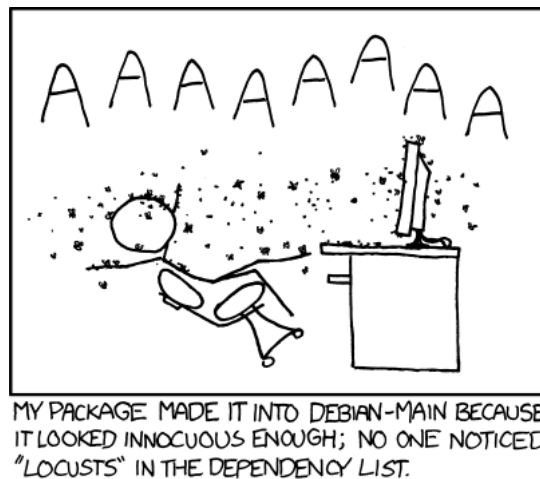


Figure 3.23: *Always check the package dependencies...*

One of the initial claims against using Linux and Unix was that software install was a nightmare. Software had been distributed in `tarballs` (`.tar.gz`) that were convenient but lacked any knowledge of system state. So you could compile source code but the code had no idea ahead of time if the proper software libraries were installed in the correct locations or if the proper versions of those libraries were installed. And each additional library had its own dependencies and those had dependencies too. You see how this could get ugly quickly. Initially there was a build system developed by a companion of Thompson and Ritchie's named Stuart Feldman; named `make`. He was also an author of the first Fortran 77 compiler, and he was part of the original group at Bell Labs that created the Unix operating system. Feldman was the Vice President of Computer Science at IBM Research. He was also Vice President, Engineering, East Coast, at Google for a time ⁴.

Feldman realized building software was difficult and created the `make` build system. The `make` system uses a file named `makefile` that includes instructions and ordered steps that can be repeated every time software is built. This allows software to be portable across systems (in theory). The utility `make` is the binary that automatically builds executable programs and libraries from source code by reading the `makefiles`⁵. Here is an example `makefile`:

```
all: helloworld

helloworld: helloworld.o
$(CC) $(CFLAGS) $(LDFLAGS) -o $@ $^

clean: FRC
rm -f helloworld helloworld.o
```

⁴https://en.wikipedia.org/wiki/Stuart_Feldman

⁵[https://en.wikipedia.org/wiki/Make_\(software\)](https://en.wikipedia.org/wiki/Make_(software))

```

# This is an explicit suffix rule. It may be omitted on systems
# that handle simple rules like this automatically.
.c.o:
$(CC) $(CFLAGS) -c $<

FRC:
.SUFFIXES: .c

```

Like many things in the Unix world, the `makefile` system has been modified and augmented but still persists as the way software project are built and installed some 30 years later. Makefiles have an arcane syntax that few people enjoy and over the years `make` has been often rewritten. In 1988 the GNU project had released their Free Software version of `make` called GNU Make or `gmake`. GNU make is included in all standard Linux distributions and is even required for compiling the Linux Kernel. There are other versions of `make` including the Unix version, `pmake` and `bmake` on the BSD Unix variants, there is a cross platform build tool called `cmake` and even Microsoft has its own build tool that can be used called `nmake`.

3.7.1 Traditional Package Managers

This style of software installation put a high barrier to who could practically use Unix/Linux. Linux distributions took to making software installation and distribution easier by introducing something initially called **Package Managers**. These were meant to eliminate all of the above process by solving two key problems. First it would solve the re-compilation of code and supporting of `make` and build tools—you technically wouldn't even need any C compiler or build tools installed. Second package managers would solve the dependency issues by keeping track of the dependency trail and be smart enough to follow that trail before installation.

3.7.1.1 .deb

The first package manager was `dpkg` which was created by Matt Welsh, Carl Streeter and Ian Murdock (founder of Debian) in 1994 as a replacement for an earlier primitive package manager. The program `dpkg` is used to install, remove, and provide information about `.deb` packages.

A Debian package (or `.deb` file) is really just made up of two tarballs ⁶. One is the control data which is listed as such:

```

Package: hello
Priority: optional
Section: devel
Installed-Size: 45
Maintainer: Adam Heath <doogie@debian.org>
Architecture: i386
Version: 1.3-16
Depends: libc6 (>= 2.1)
Description: The classic greeting, and a good example
The GNU hello program produces a familiar, friendly greeting.

```

The most important line being the **Depends** option which controls dependencies and can prevent installation if these conditions cannot be met. The second component includes the binary or pre-compiled portion of the code. Using `dpkg` is a clear step above using tarballs and compiling the code yourself.

Recently with Oracle changing the nature of how Java is supported with the transition from Java 8 and 11 and into the future, concerned companies created their own OpenJDK for download that will be supported. One such instance is [Amazon Corretto](#). These releases are created in `.deb` packages. Let's download one and install it using a simple command called `wget` or you can open a web browser and download the package and install it manually. Click on the link and save the `.deb` file on Ubuntu. Now using the `dpkg` command we can install the package manually (note there may be an error message about a missing dependency).

⁶https://www.debian.org/doc/manuals/debian-faq/ch-pkg_basics

3.7.1.2 DEB package install example

```
# From the command line terminal after you have downloaded the file
# on a Debian based system:
wget https://corretto.aws/downloads/latest/amazon-corretto-8-x64-linux-jdk.deb
sudo dpkg -i ./java-1.8.0-amazon-corretto-jdk_8.222.10-1_amd64.deb
# This method will retrieve the .rpm file directly from the internet and pass it to
# the rpm command on CentOS or Red Hat systems.
sudo rpm -iv https://corretto.aws/downloads/latest/amazon-corretto-8-x64-linux-jdk.rpm
```

You can download a .deb or .rpm file from the Vivaldi browser website: <https://vivaldi.com/download/> Vivaldi is a new browser from the team that brought us Opera browser. The packages are not available in APT or RPM, you download the .deb or .rpm. file directly and install through dpkg in Ubuntu's case. In the command below we will introduce the install command or the -i flag, which stands for **install**.

3.7.1.3 RPM Package example

```
# Download URL at: https://vivaldi.com/download/
sudo dpkg -i ./vivaldi-stable_5.3.2679.70-1_amd64.deb
sudo rpm -iv ./vivaldi-stable_5.3.2679.70-1.x86_64.rpm
# This command involves downloading and installing directly from the web
sudo rpm -i https://releases.hashicorp.com/vagrant/2.2.9/vagrant_2.2.9_x86_64.rpm
sudo dpkg -i https://releases.hashicorp.com/vagrant/2.2.9/vagrant_2.2.9_x86_64.deb
```

After executing this command on the Vivaldi packages, you will receive an error message. What is it telling you and why? You will notice that dpkg command found that it had a *dependency*, can you locate that *dependency* on <http://packages.ubuntu.com/>?

Example Usage:

```
sudo dpkg -i ./links_2.8-2_amd64.deb
```

Note that this command installs properly without any error message. <http://packages.ubuntu.com/>

Example Usage: There are other flags but the most common are these:

- sudo dpkg -r or --remove
- sudo dpkg -P or --purge
- sudo dpkg -l or --list
- sudo dpkg --status links

Example Usage: Let's use the dpkg command to list all kernel versions we have installed and the purge command to remove those old kernels entirely.

```
sudo apt-get dist-upgrade
sudo dpkg -l | grep linux-image
# x.x.-xx is the version that is not the most recent version as deleting that will
# make your system unbootable.
# uname -a will tell you the current kernel version
sudo dpkg --purge linux-image-x.x.x-xx-generic
```

```
# Sometimes there are kernel dependencies and this command will fail
# In those cases you can use the command below to remove the old kernel images
# and free space on your /boot partition.
```

```
sudo apt-get remove linux-image-x.x.x-xx-generic
```

3.7.1.4 RPM

A few years after dpkg became standard on Debian based distros, the Red Hat Linux created their own package manager out of necessity in 1998 and called it RPM (Originally Red Hat Package Manager - now known as RPM

Package Manager.) It is used across Fedora and RHEL derivatives. RPM is also used on IBM's AIX Unix distribution. RPM code and FAQ can be found at <http://rpm.org> ⁷.

Example Usage: List all installed packages: `rpm -qa`

Example Usage: List information about an installed package: `rpm -qi pkgname`

Example Usage: Install a package from a local file: `rpm -i file.rpm`

Example Usage: Remove a package from the system: `rpm -e pkgname`

Example Usage: Get information about a remote package `rpm -qpi <url>`

Similar to the previous example let us download the Vivaldi RPM and install it locally. <https://vivaldi.com/download/>. After selecting the Vivaldi 64 bit rpm and using the `-i` flag to install it, what error message is printed?

Let's try another rpm. This one is called "elrepo," it for Enterprise Linux version RHEL and CentOS. This adds up to date packages and third party repos, adding software that is not part of the stable Enterprise repos. The rpm is located at <http://elrepo.org/tiki/tiki-index.php>. You need a few pre-reqs to make this work.

```
# Add the GPG key for the repo to make sure that you are adding the official elrepo
# This works on Fedora 32+ and CentOS 8
sudo rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org
sudo yum install https://www.elrepo.org/elrepo-release-8.el8.elrepo.noarch.rpm
# If you are using CentOS 7 use this URL
sudo yum install https://www.elrepo.org/elrepo-release-7.el7.elrepo.noarch.rpm
```

3.7.1.5 Other Package Managers

As you can see that package managers were a great step forward in making Linux usable. But they don't handle the dependency issue—they don't understand the context of auto-dependency retrieval. Various solutions were created and new ones seem to pop up with each new Linux distro. Fedora based distros use yum and now use dnf (as of Fedora 23) and Debian based distros use apt. Others you might encounter are:

- [Zypper](#)
- [xbps](#)
- [Pkg for BSD](#)
- [Portage](#)
- [Pacman](#)
- [GNU GUIX](#)

3.7.2 APT

Apt (for Advanced Package Tool) is a set of core tools inside Debian. Apt makes it possible to:

- Install applications
- Remove applications
- Keep your applications up to date

The APT installer was released in 1998, the same time that Red Hat released its package manager (giving Debian a leg up and a few years head-start). APT was the out growth of a research project called *Deity* run by the Debian developers. It was planned to be a large GUI-like project, but it turns out that the APT CLI was implemented with such finesse and simplicity that all GUI plans were dropped. APT is mentioned as one of the key user based features for Debian based distros and Debian's founder Ian Murdock considers APT to be one of the best contributions of Linux ⁸. APT stands for the Advanced Packaging Toolkit. APT will interface with *dpkg* and has many similar commands but extends the functionality of *dpkg* in a critical way.

APT, which basically resolves dependency problems and retrieves the requested packages, works with *dpkg* under the hood. The main commands of APT are as follows:

- `apt-get update` – used to make sure your system is pointed to the latest repository versions. You should always run this before taking any other actions.

⁷<https://wiki.debian.org/RPM>

⁸<https://wiki.debian.org/Apt>

- `apt-get install` – used to install the application of choice
- `apt-get remove` – used to remove the application of choice
- `apt-cache search [pattern]` – used to search all your repositories for an app matching the given pattern
- `apt-get upgrade` – used to perform an upgrade of all current packages that have updates available (note in Yum this is the command `update`)
- `apt-get dist-upgrade` – this performs the same as the upgrade but will also update the kernel version and headers as well
- `do-release-upgrade` – this will update the entire distribution and move it to the next incremental version (Ubuntu 18.04 to 18.10)

3.7.2.1 Repositories

APT relies on the concept of repositories in order to find software and resolve dependencies. For apt, a repository is a directory containing packages along with an index file. This can be specified as a networked or local based location. The Debian project keeps a central repository of over 25,000 software packages ready for download and installation. This includes ability to add non-free software repositories as well. You can add additional repositories via the `add-apt-repository` command. This is used to add community maintained PPA’s—which stand for *personal package archive*. These are for packages maintained outside of Debian’s rigorous package checking standards and 1 to 2 year release window. Hence the cartoon at the beginning of the chapter.

The package system and architecture is one of the reasons for Debian’s long standing existence and credibility. The system just works. Ubuntu is a Debian derivative that utilizes this archive of packages. Remember that the founder of Ubuntu, Mark Shuttleworth, had been a Debian contributor at one point. Ubuntu builds on top of Debian’s 25,000 packages by maintaining additional *downstream* repositories that add additional software and repositories to make up the Ubuntu distribution. Ubuntu is a little more liberal on including non-free drivers for mainly high-end video cards. Users can then add additional repositories or PPA’s of their own choosing to extend APT functionality and expand that package base.

Seeing as you may want to access a more recent build of an application that may not be in the standard Debian/Ubuntu distribution or not even submitted to a repository because the version is moving too fast or the maintainer just didn’t want to package it up. For example if you want to install a newer version of the Python3 language on your system, you have to wait for another version of the OS.

Example Usage: For example, Ubuntu 2004 installs Python 3.8.x by default and doesn’t update the major version until the next Ubuntu release. But what if you needed to test a newer or older version of Python? Try the below sequence:

```
# Check your current Python3 version - 3.8.x on Ubuntu 2004
python3 -V
# Lets add the deadsnakes PPA so we can add different Python3 versions
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt-get update
sudo apt-get install python3.6 python3.9
python3 -V
python3.9 -V
python3.6 -V
```

3.7.2.2 Linux-libre PPA

There is also a PPA for Linux-libre. Linux-libre is a GNU package that is a modified version of the Linux kernel. The aim of the project is to remove from non-free or significantly obfuscated code⁹ from the Linux kernel. The downside of removing proprietary firmware from the kernel is that it will cause lose functionality for certain hardware that does not have a free software replacement available. This affects certain sound, video, TV tuner, and network (especially wireless) cards.

The resulting combination of the GNU Operating System and the kernel named Linux is the GNU+Linux operating system, although many (incorrectly) refer to it as “Linux”¹⁰.

⁹<https://en.wikipedia.org/wiki/Linux-libre>

¹⁰<https://launchpad.net/linux-libre/+archive/ubuntu/ppa>(<https://launchpad.net/linux-libre/+archive/ubuntu/ppa>)

Example Usage: Let's set our kernel free... The full instructions are at this website: <https://jxself.org/linux-libre/> Once successful, reboot your system and/while holding down shift - you should see the menu in the image below appear. Choose *Advanced Options For Ubuntu* and you will see your GNU/Libre kernels. Try to boot from one.

These are the short steps:

- To use this repository first add it to your system. Run this command:
 - `sudo apt edit-sources`
- And add the line:
 - `deb http://linux-libre.fsfla.org/pub/linux-libre/freesh/ freesh main`
- You should also fetch and install the GPG key with which the repository is signed:
 - `wget https://jxself.org/gpg.inc`
- Check that it's the right key:
 - `gpg --with-fingerprint gpg.inc`
- Make sure that you see:
 - Key fingerprint = F611 A908 FFA1 65C6 9958 4ED4 9D0D B31B 545A 3198
- As long as it matches configure the package manager to trust the key and then delete the temporary copy:
 - `sudo apt-key add gpg.inc`
- Now you will now be able to update your package manager and install Linux-libre:
 - `sudo apt update`
 - `sudo apt install linux-libre-5.5`
 - `sudo apt install linux-libre`
 - `sudo apt install linux-libre-lts`

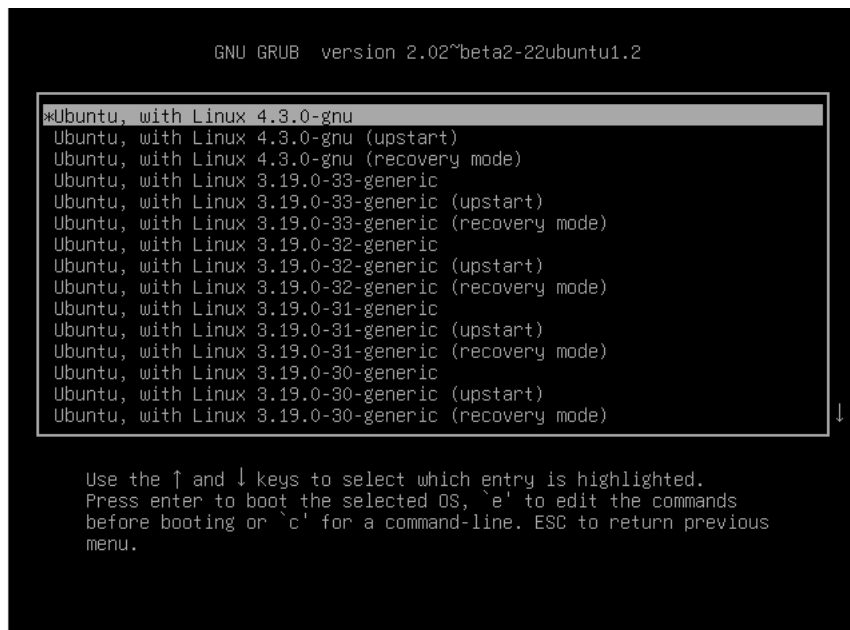


Figure 3.24: *Linux-Libre*

Here is a list of all the configuration and cache files related to APT and their location:

- `/etc/apt/sources.list`: Locations to fetch packages from.
- `/etc/apt/sources.list.d/`: Additional source list fragments.
- `/etc/apt/apt.conf`: APT configuration file.
- `/etc/apt/apt.conf.d/`: APT configuration file fragments.
- `/etc/apt/preferences`: version preferences file. This is where you would specify “pinning”, i.e. a preference to get certain packages from a separate source or from a different version of a distribution.
- `/var/cache/apt/archives/`: storage area for retrieved package files.
- `/var/cache/apt/archives/partial/`: storage area for package files in transit.
- `/var/lib/apt/lists/`: storage area for state information for each package resource specified in `sources.list`

- `/var/lib/apt/lists/partial/`: storage area for state information in transit.

3.7.3 yum & dnf

Fedora based Linux is in a bit of a transition. Its enterprise products RHEL and CentOS are still using the YUM installer. Fedora 22 and 23 still have YUM for backward support but have moved to using DNF to handle the installation of packages and dependency resolution. YUM is supported in Fedora 22 but now deprecated and DNF is the preferred installer, with YUM to be removed down the line. RPM based distros had used a tool called `up2date` prior to 2003. An opensource tool from a distro called Yellow Dog Linux lead to the creation of YUP (Yellow Dog Updater) which was then improved to become YUM (Yellow Dog Updater Modified) by the year 2003 and by 2005 every distro using RPM had moved to YUM. Yellow Dog Linux was first released in the spring of 1999 for the Apple Macintosh PowerPC-based computers and continues today as a Linux for high-end POWER7 workstations. A successor to YUM is named DNF which somehow stands for *dandified yum*. It was released in Fedora 18 and is quickly becoming the Fedora package manager of choice. YUM is still available on RHEL and CentOS but as companies move to version 7 and 8, this will begin to change too. Yum will be eventually replaced by DNF. Both YUM and DNF use repositories that are maintained by Red Hat or CentOS or even their RHEL repos.

You can find the installed repositories in `/etc/yum.repos.d`. Each file listed will contain information about the URL where it retrieves repos. There is also an ability to set priorities as to which repo is checked first. As we did in previous chapters, we added RPM repos. The most famous package for adding additional software is RPMForge, <http://rpmfusion.org/>. Taken directly from their website, “*RPMFusion ships packages that Fedora and Red Hat don't want to ship standard with their distro.*” This includes free software as well as non-free software that cannot be shipped due to the GPL nature of Fedora.

```
[vagrant@fedora28 ~]$ ls /etc/yum.repos.d/
fedora-cisco-openh264.repo fedora.repo fedora-updates.repo fedora-updates-testing.repo
[vagrant@fedora28 ~]$
```

Figure 3.25: *Installed Repositories Fedora 30*

`FFmpeg` is a free software project, the product of which is a vast software suite of libraries and programs for handling video, audio, and other multimedia files and streams. If we try to install it via `sudo dnf install ffmpeg` we get this message, why?

```
Last metadata expiration check performed 0:45:17 ago on Tue Nov 10 20:42:33 2015
.
No package frodo available.
Error: Unable to find a match.
```

Figure 3.26: *Unable to Find a Match*

First we want to check if we have the correct RPM name. We can search through our repos looking for the name by typing the `sudo dnf search [fF]mpeg*` command. This will return two results—the package and a related dependency and watch out, RPM also tends to be case-sensitive.

To enable the download the RPMFusion repos for adding additional software of free and non-free type you can type the following commands:

```
sudo dnf install http://download1.rpmfusion.org/free/fedora/\
rpmfusion-free-release-$(rpm -E %fedora).noarch.rpm

sudo dnf install http://download1.rpmfusion.org/nonfree/fedora/\
rpmfusion-nonfree-release-$(rpm -E %fedora).noarch.rpm
```

Note for RHEL/CentOS the installation URL is slightly different:

```
# the "\" is a line continuation character to extend a single line beyond for
# readability purposes
# For CentOS 7
sudo yum localinstall --nogpgcheck \
```



```

http://download1.rpmfusion.org/free/el/updates/7/x86_64/\
r/rpmfusion-free-release-7-4.noarch.rpm
# For CentOS 8
sudo yum localinstall --nogpgcheck \
http://download1.rpmfusion.org/free/el/updates/8\
/x86_64/r/rpmfusion-free-release-8-0.1.noarch.rpm
# Fedora 36
sudo yum localinstall --nogpgcheck \
http://download1.rpmfusion.org/free/fedora/\
rpmfusion-free-release-32.noarch.rpm

```

If you are using CentOS or RHEL you need to first install the **EL-Repo** before the RPMFusion, but not for Fedora. No it isn't Spanish for “*the repo*”, but stands for Enterprise Linux Repo—located at <http://elrepo.org/tiki/tiki-index.php>. The ELRepo Project focuses on hardware related packages to enhance your experience with Enterprise Linux. This includes filesystem drivers, graphics drivers, network drivers, sound drivers, webcam and video drivers. This book will not focus on the RHEL update and RPM repos but I wanted to make you aware of it.

Once those RPMFusion repos have been added you can now retry the example above and install **FFMpeg**, **Denyhosts**, and **Links**. Unlike Ubuntu and Debian where we need to update the repositories - DNF and YUM will auto handle that.

Example Usage: You can install additional packages now that you have the RPMFusion repos added. Try to install links the web browser that failed when we tried to install it. The command is `sudo dnf install links`. The command `sudo dnf remove links` will uninstall it. The command `sudo dnf upgrade` will upgrade all packages that have updates pending. You can now use DNF to [upgrade your system](#) as well. These are the series of commands to install the DNF upgrade plugin and then execute the process.

- `sudo dnf update --refresh`
- `sudo dnf install dnf-plugin-system-upgrade`
- `sudo dnf system-upgrade download --releasever=32`
- `sudo dnf system-upgrade reboot`

3.8 New Package Managers as App Stores

The one thing that you can say the mobile revolution brought into computing is the concept of an “app store.” Apple obviously came first, Google Play Store, and even the Microsoft Store took root. The concept of an app store is pretty ubiquitous at this point. These platforms are not based on free and opensource, they are called “walled gardens,” because you are free to install any software as long as it comes from the curated app store. On a fundamental level Linux is conceptually not compatible with the concept of an app store. But the concept that an app store provides, controlling/standardizing software versions, ease of install/remove, and basic sandbox security technology for apps—the benefits cannot be overlooked in regards to usability. To this end two standardized methods were developed: Flatpak and snaps. Snaps were created by Ubuntu and is account based across Linux distros that support the snapd library. Flatpak is the competing standard championed by Red Hat.

3.8.1 Snaps and snapd

In 2019 Canonical, the parent company of Ubuntu, introduced a new package manager architecture called Snaps. This architecture includes a package manager process called **snapd**, packages called **snaps**. The technology is housed at a neutral organization site called [snapcraft](#).

Snaps are an attempt to create a similar concept to modern App Stores on mobile devices. They allow for updating packages in place and the packages contain all needed dependencies. This is a different focus from simple package managers like APT and RPM. In addition, you can “package” existing applications into Snaps and or distribute them via a Snap account to devices and then instantly download them on another machine that supports snaps. Currently, Ubuntu distributions are the only Linux distros that have Snappy installed by default.

Snaps, similar to the App Store concept, allows you to:

- Simple to package leveraging your existing tools
- Automatic updates for everyone

- Reach tens of millions of Linux systems
- Roll back versions effortlessly
- Integrate easily with build and CI infrastructure
- Free for open and closed source projects

Since Ubuntu Desktop 20.04, some packages when installed via apt, have been redirected to use the appropriate Snap package, which has bothered some, and one distro, Linux Mint, has removed Snaps all together. Snaps can be installed from the commandline or via the Store Icon in the Ubuntu Desktop.

You can open the Ubuntu Software Icon on the Favorites Bar and select some software. Lets install Notepad++. You can also install Snaps from the command line. Open a terminal and issue the below commands to find software package names (since there is no GUI to browse):

```
# This command will show your installed Notepad++ snap from the store
sudo snap list
sudo snap find vscode
# You will see a list of output and a column with the term "classic" or not.
# This means that the first version of snap that packages were built for
# did not have the sandboxing technology of a traditional app.
# They are conventional application packages.

# This will install Visual Studio Code
sudo snap install vscode --classic
# To install a note taking app that is not a classic snap
sudo snap install simplenote
# To see all snaps installed
sudo snap list
```

3.8.2 Flatpak

Supported on all Linux operating systems and functions like an app store – where you can install, remove, and update packages all from a single command - [Flatpak](#). You can see software available at [Flathub](#).

Flatpak essentially connects the concepts of repositories and app packages, but the drawback is that Flatpak manages its own applications and list of installed applications. It is installed as standard on Fedora 32, but not other distributions. Let's install some Flatpak and some applications: The first thing is to install the Flatpak package. [That is done at the Flatpak repo page](#). Let's try it with an Ubuntu distribution:

3.8.2.1 Install Flatpak on Fedora/CentOS

Flatpak is installed by default in Fedora 32+ and Centos 7+. You just need to add the flatpak repo:

```
flatpak remote-add --if-not-exists flathub https://flathub.org/repo/flathub.flatpakrepo
```

3.8.2.2 Installing Flatpaks on Fedora/Ubuntu/CentOS

Once that is done here is an example of installing Flatpaks:

- `sudo flatpak install flathub org.videolan.VLC`
- `sudo flatpak install flathub com.discordapp.Discord`
- `sudo flatpak install flathub io.atom.Atom`
- `sudo flatpak install flathub org.blender.Blender`
- `sudo flatpak install flathub com.obsproject.Studio`

These apps appear on your start menu after a logout and log back in. They can also be launched via the Flatpak command from the commandline. Note that from the command line you don't run these as `sudo`. You can find the official Flatpak name by issuing the command: `Flatpak list`.

- `flatpak run flathub org.videolan.VLC`
- `flatpak run flathub com.discordapp.Discord`
- `flatpak run flathub io.atom.Atom`
- `flatpak run flathub org.blender.Blender`

- `flatpak run flathub com.obsproject.Studio`

3.8.3 AppImage

A third new package manager has arisen called [AppImage](#). The idea here is for standalone compatibility. AppImages are larger files because they contain all of the necessary libraries to run, packed with the application. This is sometimes called a fat-binary. This eliminates worrying about software library versions as everything is self-contained. This is also a good way to distribute older software that may not run on newer systems. AppImage should have vendor support, but currently doesn't have wide deployment, but the AppImage is supported across most Linux distros due to third party work. AppImages are self-contained binaries so there is no main store or tool for updating, you have to find them and keep an eye on them to make sure they are the latest or most up to date versions.

AppImageHub

- [AppImage for Visual Studio Code](#)
- [AppImage for Audio/Video Player](#)
- [AppImage for Calibre e-book manager](#)
- [2D Animation Studio](#)

3.9 Compiling and Installing source code

In addition to packages you may still want to compile software from source. This way you can take advantage of the latest compiler optimizations and CPU support. Or compile older versions that have a feature you need that is no longer supported as a package any more.

3.9.1 GNU GCC

The main tool needed is the GNU C compiler or GCC for short. This was one of the first items that Richard Stallman created in the GNU project and to this day is needed for building the Linux Kernel and is the standard build tool for Free Software. There are competing software stacks and compilers, as of version 10 the FreeBSD project deprecated GCC and chose the [Clang](#) project, originally designed by Apple to support [Xcode](#), instead. Apple abandoned the GCC compiler because of the restrictions placed on it by GPLv3, which is an interesting side effect of GPLv3. The GCC compiler has grown to include other languages over the years as well. You can install the GCC compiler and all the additional build tools in Debian/Ubuntu by typing: `sudo apt-get build-essential`. In Fedora you would add these two commands via yum or dnf: `sudo yum groupinstall 'Development Tools'` and `sudo yum groupinstall 'Development Libraries'`. You can compile code directly by invoking the `gcc` or `g++` command.

3.9.2 GNU Make

As mentioned prior the GNU make command is used to actually compile the C code and all the directives stated in the build file. That compiled source is then placed into the proper system directories by the `make install` command. This command needs *superuser* privileges to move files to directories not owned by the user, but the `make` command doesn't need sudo—resist the temptation! The `--prefix=` is the default location where you want to store the compiled Apache2 binaries, it defaults to `/usr/local/apache2/`.

Let's compile something to see how this works. This link is to the Apache webserver version 2.4.x latest source code: <http://httpd.apache.org/docs/2.4/install.html>. Let's install some pre-requisites:

```
# Pre-reqs needed first -- assuming Ubuntu 20.04
sudo apt-get install build-essential libapr1 libapr1-dev libaprutil1 \
libaprutil1-dev libpcre3 libpcre3-dev
# Command to retrieve the source code
wget http://www.gtlib.gatech.edu/pub/apache/httpd/httpd-2.4.54.tar.gz
# Command to unzip the source code
tar -xvzf httpd-2.4.54.tar.gz
# command to change directory to extracted source code
cd httpd-2.4.54
# commands to build
./configure
```

```
make
sudo make install
sudo /usr/local/apache2/bin/apachectl -k start
```

Now open a web browser and navigate to <http://127.0.0.1> and you should see the message: “It Works!”

3.9.3 Using Python to Install Python Based Programs

Python has its own package installer called **pip** which allows for software that is written in Python and independent of any Linux package manager to be installed. Pip allows you to install newer versions of a Python package without having to wait for a Linux distro’s package manager.

You can see an example of how to install Python language packages (eggs). A popular package is called **OpenCV**, which allow you to do computer vision, object recognition, and image manipulation. We can install this very useful package via **pip**. We can do this by opening the Terminal via the **Activities** tab and run these commands:

```
# command to install python3 pip
sudo apt-get install python3-setuptools python3-pip python3-dev
python3 -m pip install opencv-python
```

There is sample code in the textbook repository. Under the directory **files > Chapter-03** and run the command:

```
python3 show-image.py
```

3.9.4 Installing VirtualBox Guest Additions 7.x

You may have noticed that when a guest VM is successfully installed the screen resolution may be very small and the mouse integration features are not working. By default VirtualBox doesn’t know what your host system’s underlying hardware is. So it guesses by providing a lowest common denominator set of hardware drivers, usually for pretty old, but well known set of hardware. In order to install higher quality drivers to enable more features VirtualBox provides something called “*guest additions*” to enable exclusive features that are not normally available in an operating system. These features include things such as shared folders, cut and paste support, and even support for multiple monitors and higher resolutions.

VirtualBox Guest Additions can be installed by selecting an installed virtual machine and selecting the menu item under **DEVICES** then select the “Insert Guest Additions CD Image.” For Windows and Mac as the guest VM OS this is a pretty straight forward install - the attached Guest Additions ISO appears within the VM and you simply double click it and run through the menu, reboot, and new features are added. For Linux you need to compile these extensions into the kernel and some extra tools are needed.

“The Guest Additions are designed to be installed inside a virtual machine after the guest operating system has been installed. They consist of device drivers and system applications that optimize the guest operating system for better performance and usability.” <https://www.virtualbox.org/manual/ch04.html>

The Guest Additions offer the following features:

- Mouse pointer integration
- Shared folders
- Better video support
 - Finally, if the Guest Additions are installed, 3D graphics and 2D video for guest applications can be accelerated; see Section 4.5, “Hardware-accelerated graphics”
- Time synchronization
- Shared clipboard
- Automated logons (credentials passing)

3.9.4.1 Ubuntu 22.04 and 23.04 Desktop and Server

```
# Assuming you are using VirtualBox 7.x and you have inserted
# the VirtualBox Guest editons iso (under Devices)
sudo apt-get update
sudo apt-get install build-essential dkms linux-headers-$(uname -r)
sudo mount /dev/cdrom /media
```

```
cd /media
sudo ./VBoxLinuxAdditions.run
sudo reboot
```

3.9.4.2 Ubuntu 20.04 and 22.04 Server

```
# Assuming you are using VirtualBox 7.x and you have inserted
# the VirtualBox Guest editons iso (under Devices)
sudo apt-get update
sudo apt-get install build-essential dkms linux-headers-$(uname -r)
sudo mount /dev/sr0 /media
cd /media
sudo ./VBoxLinuxAdditions.run
sudo reboot
```

3.9.4.3 Debian 11 and 12

```
# Assuming you are using VirtualBox 7.x and you have inserted
# the VirtualBox Guest editons iso (under Devices)
su - root
apt-get update
apt-get install build-essential dkms linux-headers-$(uname -r)
mount -r /dev/cdrom /media/cdrom
cd /media/cdrom
./VBoxLinuxAdditions.run
reboot
```

3.9.4.4 Red Hat based Fedora 37/38, AlmaLinux 9, Rocky Linux 9

```
# Assuming you are using VirtualBox 7.x and you have inserted
# the VirtualBox Guest editons iso (under Devices)
sudo dnf update kernel*
sudo reboot
sudo dnf install -y gcc gcc-c++ kernel-devel kernel-headers make bzip2 perl
sudo mount -r /dev/cdrom /media
cd /media
sudo ./VBoxLinuxAdditions.run
sudo reboot
```

3.9.4.5 Manjaro and Arch Linux

```
# Assuming you are using VirtualBox 7.x and you have inserted
# the VirtualBox Guest editons iso (under Devices)
 #(find your kernel version)
sudo mhwd-kernel -li
# XX is your kernel version
sudo sudo pacman -S linuxXX-headers
sudo mount -r /dev/cdrom /mnt
cd /mnt
sudo ./VBoxLinuxAdditions.run
sudo reboot
```

11

If successful you can reboot the Linux guest VM and you will notice the changes take place immediately. If some of these commands are not familiar that is ok - we will cover them all in later chapters. Without these additional tools installed you will receive an error message similar to this:

¹¹How to install Virtual Box Guest Additions on Manjaro Linux

3.9.5 VirtualBox Features

If you are using Windows, Mac, or Linux you need to download the appropriate version from the VirtualBox homepage. Version 6.x.x is the [current version](#).

Feature List for VirtualBox

- Guest multiprocessing (SMP)
- USB device support
- Seamless windowing
- Shared folders
- Hardware compatibility
- Full ACPI support
- Multiscreen resolutions
- Built-in iSCSI support
- PXE Network boot
- Remote machine display
- Video and screenshot capture within virtual machines

3.10 Chapter Conclusions and Review

Through this chapter we gained an understanding of what x86-based virtualization does. We learned about the purpose of a hypervisor and how opensource tools such as VirtualBox provide these services. We learned how to install Ubuntu and Fedora based distros in the most common scenarios. We learned about VirtualBox features and how to install Linux.

3.10.1 Review Questions

- 1) What is the term for the industry standard file format that is used to install a Linux distro?
 - a. ISO
 - b. ZIP
 - c. Disk-ISO
 - d. Distro
- 2) What is currently the most common Linux install media type?
 - a. CD-ROMs
 - b. Network based installs
 - c. USB
 - d. Thunderbolt
- 3) What is the name of recommended tool used to create bootable Linux install media?
 - a. Pendrive Linux
 - b. etcher.io
 - c. UNetbootin
 - d. Image Magick
- 4) What is the technology that is inserted between ring 1 and ring 0 that enables virtualization?
- 5) The operating system that the hypervisor resides on is called the _____ system?
- 6) Hosted or desktop virtualization is called what type of hypervisor?
- 7) Bare Metal or Native Virtualization is called what type of hypervisor?
- 8) Each Linux installation distro provides a mechanism to compare what you downloaded with what you expected to download, what is that called?
 - a. mount point

- b. checksum
 - c. receipt
 - d. mdsum
- 9) What is the name of the driver package you can install in VirtualBox in order to enable features such as shared clipboard, larger screen resolution, and mouse pointer integration?
- a. Kernel modules
 - b. Kernel drivers
 - c. VirtualBox extensions
 - d. ISO extensions
- 10) What is the name for a Linux distribution that runs in memory?
- a. Rapid CD
 - b. Live ISO
 - c. Install Disk
 - d. Trick question
- 11) What feature doesn't dpkg handle/support?
- a. Removing software
 - b. Installing dependencies
 - c. Versioning
 - d. Author Information
- 12) What is the APT command to add an additional software repository in Ubuntu/Debian, named: `ppa:linux-libre/ppa`, to your APT system?
- a. `sudo add-repository ppa:linux-libre/ppa`
 - b. `sudo add-apt-repository ppa:linux-libre/ppa`
 - c. `sudo apt-add-repository ppa:linux-libre/ppa`
 - d. `sudo apt-add ppa:linux-libre/ppa`
- 13) Which distro(s) supports the .deb package?
- a. Ubuntu only
 - b. Debian Family
 - c. Debian and Red Hat
 - d. None of the above
- 14) Which distro(s) supports the RPM package?
- a. CentOS only
 - b. Red Hat Family
 - c. Debian and Red Hat
 - d. None of the above
- 15) We talked about using GCC to compile and install software, what was the other language/package manager discussed in the chapter?
- a. G++
 - b. APT
 - c. Python
 - d. None of the above
- 16) Describe the purpose of VirtualBox Guest Additions?
- 17) What is the RPM command to install a package from the command line?
- a. `rpm -qa *.rpm`
 - b. `rpm install *.rpm`
 - c. `rpm -q *.rpm`
 - d. `rpm -i *.rpm`

- 18) After building software from source and running the `./configure` command, what is the next step?
- Run the `make install` command
 - Run the `sudo make install` command
 - Run the `install` command
 - Run the `make` command
- 19) What is the name of the new package managers developed by Canonical and Red Hat?
- Flatpak and apt
 - Flatpak and snap
 - snappy and flatter
 - dnf and apt
- 20) What is the DNF command used to install additional software repositories? Use this URL to an RPM:
http://download1.rpmfusion.org/free/el/updates/7/x86_64/r/rpmfusion-free-release-7-4.noarch.rpm
- `sudo dnf install repo http://download1.rpmfusion.org/free/el/updates/7/x86_64/r/rpmfusion-free-release-7-4.noarch.rpm`
 - `sudo dnf http://download1.rpmfusion.org/free/el/updates/7/x86_64/r/rpmfusion-free-release-7-4.noarch.rpm`
 - `sudo dnf install http://download1.rpmfusion.org/free/el/updates/7/x86_64/r/rpmfusion-free-release-7-4.noarch.rpm`
 - `sudo install http://download1.rpmfusion.org/free/el/updates/7/x86_64/r/rpmfusion-free-release-7-4.noarch.rpm`

3.10.2 Podcast Questions

Listen/watch the FLOSS podcast number 130 with the [VirtualBox Developers - http://twit.tv/floss/130](http://twit.tv/floss/130)

- ~2:35 Who is Andy Hall and Achim Hasenmuller?
- ~3:00 What is Simon Phipps relationship to the VirtualBox project?
- ~4:45 What does VirtualBox do in Andy Hall's words?
- ~6:00 About what year did the product that became VirtualBox start?
- ~11:20 According to Simon, what is the definition of open core?
- ~14:17 How does VirtualBox fit into Oracle's business model?
- ~16:15 As of the time of the podcast (2010) how many downloads did VirtualBox have?
- ~20:25 How does VirtualBox handle virtualized I/O?
- ~22:40 What did Intel and AMD introduce to help ease virtualization in VirtualBox?
- ~26:00 What two models of network card did VirtualBox choose to represent their virtual hardware and why?
- ~27:40 What does VirtualBox almost get native performance on?
- ~29:29 How does VirtualBox treat USB devices in Guest OSes?
- ~31:00 What are 4 virtual networking modes in VirtualBox?
- ~32:30 What is the difference between NAT and Bridged networking?
- ~39:30 What Type of hypervisor is VirtualBox?
- ~51:30 Why can't you virtualize Mac OSX on VirtualBox (as of 2014)?

3.10.3 Lab

You will need to do some research and find the download links for the Linux and BSD based distros below and install them in VirtualBox. You will need to install the latest version of [VirtualBox 6.1.x](#) in order to complete this exercise; it can be installed via Chocolatey or Brew package managers as well. If you are using an M1 Mac, you will need to purchase a copy of a comparable software called [Parallels Virtualization for M1 Macs](#).

Complete each install to disk—there should NOT be an INSTALL ICON on the desktop – your screenshot is taken after a complete install is finished and a reboot has taken place. Assume each instance listed below is 64-bit version. Take a screen shot of each desktop after logging in. There are 17 different distributions listed. If a version number is not listed, assume the latest version unless noted.

- Debian Based
 - Ubuntu 20.04 Desktop edition
 - Lubuntu 20.04 Desktop edition
 - XUbuntu 20.04 Desktop edition
 - Ubuntu 20.04 Server edition
 - Trisquel Linux

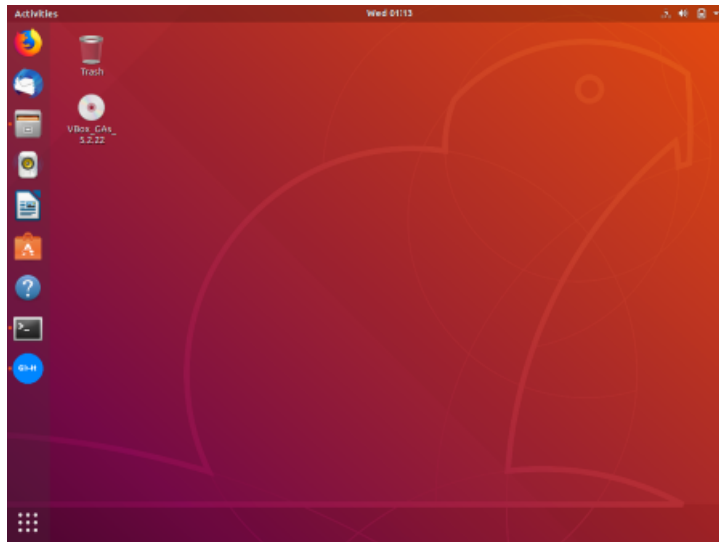


Figure 3.27: *Sample Deliverable*

- Solus Linux - MATE
- PureOS
- Red Hat Based
 - Fedora 35 - Workstation edition
 - CentOS Stream - Minimal install
- Illumos / Solaris Based
 - OmniOS Community Edition
- BSD based
 - FreeBSD
- Linux
 - Alpine Linux - Standard
 - MX Linux
 - Pop!_OS
 - Linux Mint
- Network Based Install
 - openSUSE Leap
 - Debian 11.x
- Installing Guest Additions in each VirtualMachine
 - Following the instructions in section 3.9.5, install the VirtualBox Guest Additions Package in the below listed Operating systems, placing a screenshot directly below each bullet point
 - * To show the successful install of the guest additions, maximize the screen and take the screenshot showing the installed guest additions. These are the example before and after maximized images:

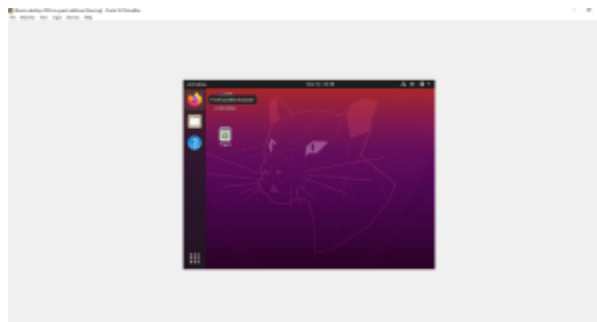


Figure 3.28: Before Guest Additions

*

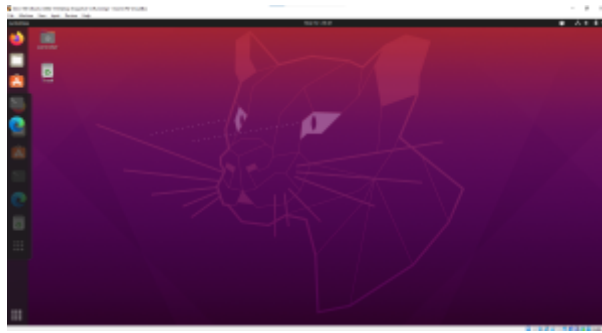


Figure 3.29: After Guest Additions

- *
 - Ubuntu Desktop
 - Fedora Desktop
 - Debian
 - Pop!_OS
 - Ubuntu Server

3.10.3.1 Footnotes

Chapter 4

Desktop Linux and GUIs



Figure 4.1: *GUIs???* NIH

4.1 Objectives

- Learn about the different types of desktop Linux
- Understand the nature and use of X, Wayland, and other GUI compositors
- Understand the purpose of window managers and the nature of desktop environments
- Know the major features of GNOME desktop environments
- Understand GUI package managers and application stores
- Understand theming on the Linux Desktop

4.2 Outcomes

At the conclusion of this chapter you will have a strong knowledge of the X Windows system and why it is the foundation of Desktop Linux. You will be able to implement various window managers and desktop environments, selecting the proper environment to utilize and suit your hardware needs. You will be comfortable using and recognizing the features of GNOME desktop environment.

4.3 From Paper Tape to CLI to GUIs to 4K

When you think about how we use computers today the one thing we take for granted is a *GUI*, **Graphical User Interface**. The GUI is intricately tied to the way computers are used, why do think Microsoft calls its operating system Windows? Unix however, had its start in the late 1960s and 1970s—there was vastly different technology

available at that time. The PDP-7s that Thompson and Ritchie used were called *Teletypes* or *TTYs* that had punch card readers for data entry and paper tape for output connected to a PDP-7 via a modem. You will notice that even today in Unix screen outputs are still referred to as TTYs. By the mid 1970s we begin to see what are called *dumb terminals* appear which could be considered primitive monitors.

The **Datapoint 3300** was one of the more common models representing the era. It supported control codes to move the cursor up, down, left and right, to the top left of the screen, or to the start of the bottom line and displayed a whopping 72 by 25 rows of characters in the days before microprocessors and ram were part of terminals. By 1978 Digital Equipment Corporation (DEC) had released something akin to an upgrade for the dumb terminal called a *smart terminal*.

The **DEC VT-100** was the most popular model: it had character and cursor positioning and was connected to the Unix system via serial ports and a modem. The VT-100 set the standard for 80 by 24 rows and columns still in use today. These terminals had the backing of DRAM and an Intel 8080 processor. Open up any terminal emulator in Linux and see its default size. Open up cmd.exe in Windows and you will notice the same default dimensions. Why? Sociologists always talk about nature vs. nurture. Nature plays a huge part in software development. All the developers who built Terminal emulators and cmd.exe for Windows “grew up” using Unix systems on a DEC VT-100 screen in college and at work. It was a natural fit. The VT-100 and VT-220 continued with wide deployment and success but the concept of color or being able to draw any shapes to the screen was in the hands of a few expensive and proprietary companies; everything was still **ASCII** or character based.

4.3.1 VT-100



This screenshot is a file listing from the `/dev` (devices) folder on a Fedora 22 system

```
/dev/tty /dev/tty19 /dev/tty3 /dev/tty40 /dev/tty51 /dev/tty62
/dev/tty0 /dev/tty2 /dev/tty30 /dev/tty41 /dev/tty52 /dev/tty63
/dev/tty1 /dev/tty20 /dev/tty31 /dev/tty42 /dev/tty53 /dev/tty7
```

Figure 4.2: *TTY Alive Today*

4.3.2 Virtual Consoles

With the X system the idea of having discrete terminals went away. Now you could have multiple *virtual terminals* on one system that emulated the features of a DEC VT-100 or VT-220 terminal. If you hit the `ctrl + alt + F1-F7` within your Linux distro you will jump to 1 of 7 different virtual terminals enabled by default. Usually F7 is the default GUI but it can vary from operating system to operating system.

¹By Jason Scott (Flickr: IMG_9976) CC BY 2.0, via Wikimedia Commons

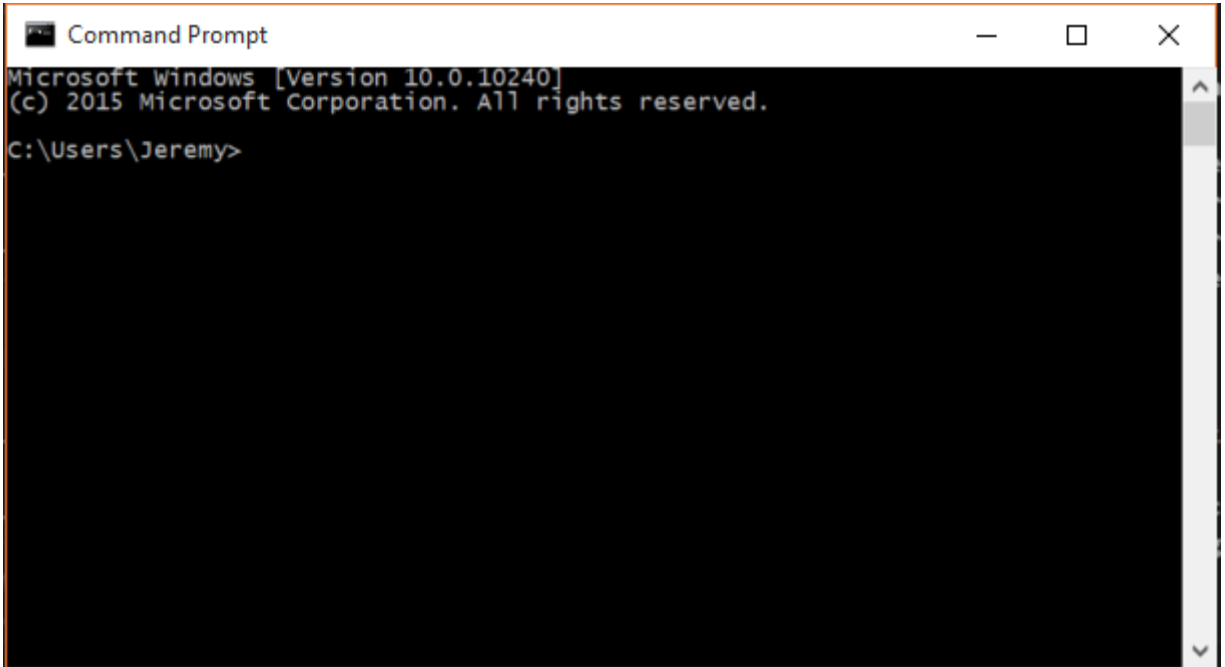


Figure 4.3: *Windows cmd.exe eerily similar to the VT-100*

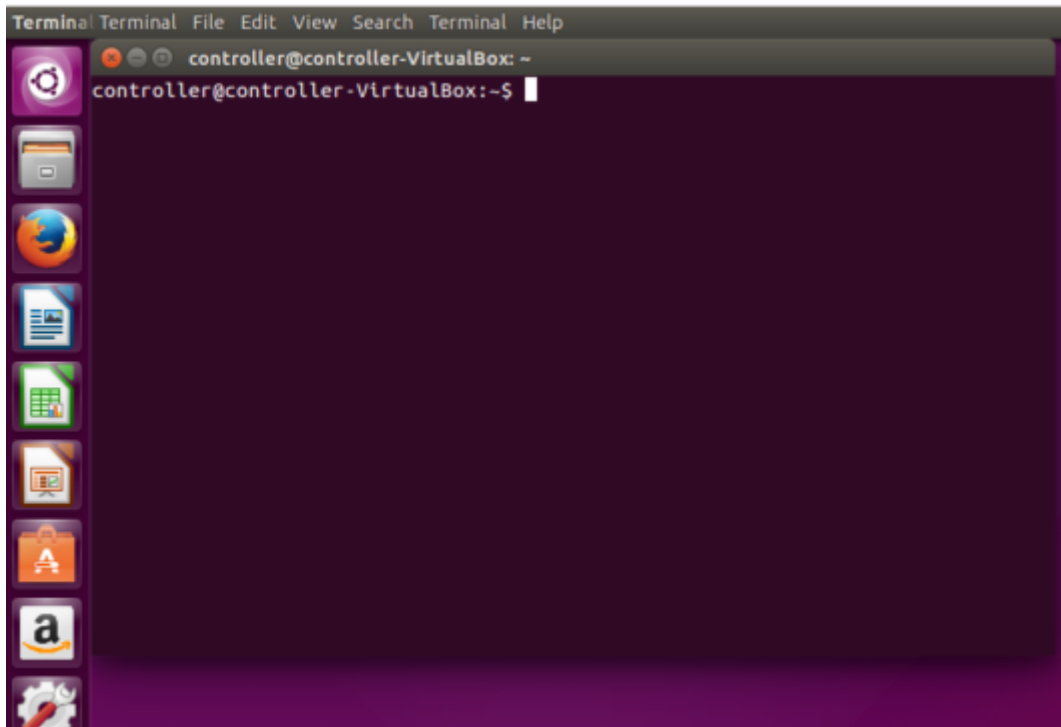


Figure 4.4: *Ubuntu's GNOME Terminal emulates the VT-100 directly*

4.4 Along Comes an X

Beyond the simple green screen terminals by the mid 1980s we began to see proprietary primitive UNIX GUIs from commercial vendors, especially from Sun. These GUI interfaces were placed on high-end workstations sold at premium prices for the day. One may wonder why you would even bother to create a GUI in the first place. Computing at that time was beginning to take less of an iterative/batch processing approach to computing and a more of an interactive approach. GUIs were in the hand of the privileged few vendors but not of anyone else. By 1985 MIT open sourced a project called X. This project was after Richard Stallman had left the school but continued in his spirit. X was a continuation of a primitive GUI called “W”. As Unix originally had no concept of a GUI, one had to be bolted on. But the spirit of X was not to enforce any standards on the user.

X uses a client–server model: an X server communicates with various client programs. The server accepts requests for graphical output (windows) and sends back user input (from keyboard, mouse, or touchscreen). The server may function as:

- An application displaying to a window of another display system
- A system program controlling the video output of a PC
- A dedicated piece of hardware.

4.4.1 X

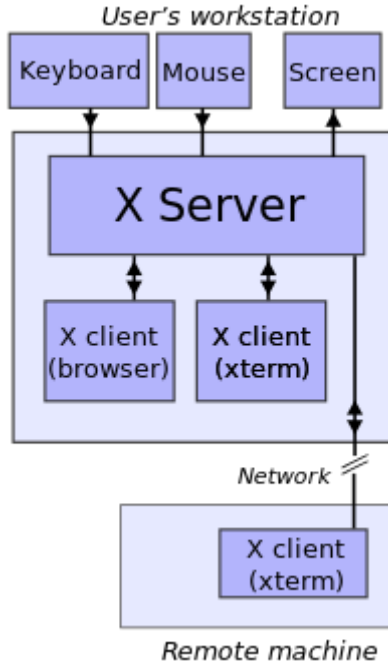


Figure 4.5: X Diagram

Even in the mid 1980s the dream of having your own desktop PC was a bit of a ways off. Large single Unix systems were still the nature of computing so the X system fit in, especially in the idea of remote terminals or underpowered system connecting remotely to a central server and running their own *X Window* remotely. X is often misnamed as X Windows, but the name for the project is “X”.

“...in 1987-1988 X established itself as the standard graphics engine for Unix machines, a hardware-independent neutral platform on top of which to build GUIs” Eric S. Raymond writings.

“X provides the basic framework for a GUI environment: drawing and moving windows on the display device and interacting with a mouse and keyboard. X does not mandate the user interface — this is handled by individual programs.” https://en.wikipedia.org/wiki/X_Window_System

By 1987 version 11 of the protocol had been released, being commonly referred to as X11. The industry wanted to prevent fragmentation and they asked MIT to form a steering consortium for the project. The MIT X Consortium

became the X Consortium Inc. in 1993. By 1996 the X Consortium, with vendor input, released one of the first standard Unix GUIs called the **CDE** *Common Desktop Environment*. By 1997 the X Consortium transformed into the Open Group. The Open Group angered many people by relicensing the project under stricter commercial terms in an effort to make some money to support the project—but this action backfired. By 1999 the Open Group had ceased functioning and ceased really developing X. The Open Group created *X.Org* to maintain the X11 standard.

At the same time a parallel implementation base on X11R5 was being developed by IBM for their PCs. The project was started in 1992 as XFree86. This code was donated to the X Consortium and the code has been part of the larger code base that the X Consortium managed. So while the X Consortium/Open Group/[X.Org](#) was in charge of X development, they actually weren't innovating or developing anything. It was the group at XFree86 that was doing all the innovation.

The story gets more complicated. As the userbase of Linux surges in the early 2000s as desktop Linux and GUIs grow, X use would naturally follow that growth curve as well. You have two problems: [X.Org](#), a vendor controlled stewardship organization, was not doing any development on X, and XFree86 was starting to narrow its *openness* just like the OpenGroup did in 1997, expelling some of the original X creators who had joined the XFree86 project and changing open licenses to be non-GPL compliant (non-Linux compatible license).

This led to the founding of the X.Org Foundation merging the two groups. This was a breakthrough event. It basically reconciled the two groups under one new foundation, [X.Org Foundation](#), it also ensured that the creators of the X project were once again the stewards, and finally it insured that there was a GPL based and protected X project that developers and companies could contribute to as well. The X.Org Foundation continues to innovate and foster the project to this day.

As X was being developed, the most breakthrough commercial GUI available in the early 80s was released by Apple in 1984². Here is a [link to an article](#) from the developers about how it came to be.



Figure 4.6: *Apple Macintosh 1984*

X has a definite advantage in that it is very mature and very stable for better or worse. In the diagram earlier in the chapter you can see one of X's major faults. It was designed not with a desktop GUI in mind, those didn't exist at the time. Every desktop element is a client that has to make calls to the X server in order to render any changes to the screen. This adds extra layers of overhead and also becomes a [security issue](#) with an X client being able to connect unauthenticated in some cases to other X servers remotely on the network.

²<http://www.allaboutapple.com/> CC BY-SA 2.5 it, CC BY-SA 2.5 it, GFDL or CC-BY-SA-3.0, via Wikimedia Commons

4.4.2 Project Wayland

In the late part of the 2000s, then Red Hat engineer Kristian Høgsberg, began to reimagine the nature of the Linux GUI compositor with a simpler desktop driven protocol. With the help of other senior X.Org developers they began to create a successor project to the X project. By 2012 they had their first code release usable for beta testing. This project was called [Wayland](#). The famous architect of the City of Chicago Daniel Burnham once said, “*Make no little plans. They have no magic to stir men’s blood and probably will not themselves be realized.*” Every single desktop environment and window manager runs on X—this means rewriting or extending every single graphical toolkit and graphical development environment out there. The Wayland project had that just in mind. What is Wayland then?

“Wayland is a protocol for a compositor to talk to its clients as well as a C library implementation of that protocol. The compositor can be a standalone display server running on Linux kernel modesetting and evdev input devices, an X application, or a wayland client itself. The clients can be traditional applications, X servers (rootless or fullscreen) or other display servers.” [Wayland Project](#)

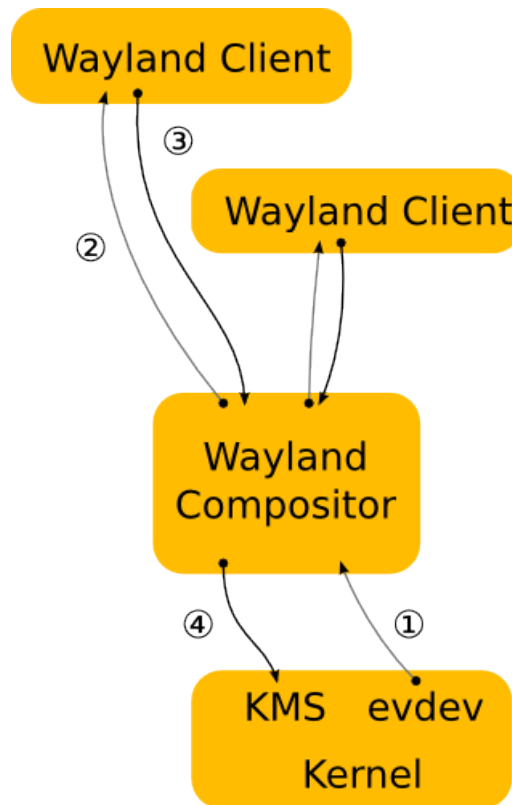


Figure 4.7: *Wayland Rendering Model*

- The kernel gets an event and sends it to the compositor. This is similar to the X case, which is great, since we get to reuse all the input drivers in the kernel.
- The compositor looks through its scenegraph to determine which window should receive the event. The scenegraph corresponds to what’s on screen and the compositor understands the transformations that it may have applied to the elements in the scenegraph. Thus, the compositor can pick the right window and transform the screen coordinates to window local coordinates, by applying the inverse transformations. The types of transformation that can be applied to a window is only restricted to what the compositor can do, as long as it can compute the inverse transformation for the input events.
- As in the X case, when the client receives the event, it updates the UI in response. But in the Wayland case, the rendering happens in the client, and the client just sends a request to the compositor to indicate the region that was updated.
- The compositor collects damage requests from its clients and then recomposites the screen. The compositor can then directly issue an ioctl to schedule a pageflip with KMS.

At one time it was thought that X was too deep into the bones of Linux and may never be ever fully replaced. But we are seeing massive amounts of work from industry to make Wayland a reality. With Intel, Nvidia, the X.Org

Foundation and Red Hat leading the way, Fedora 25 was one of the first distros to run Wayland natively.

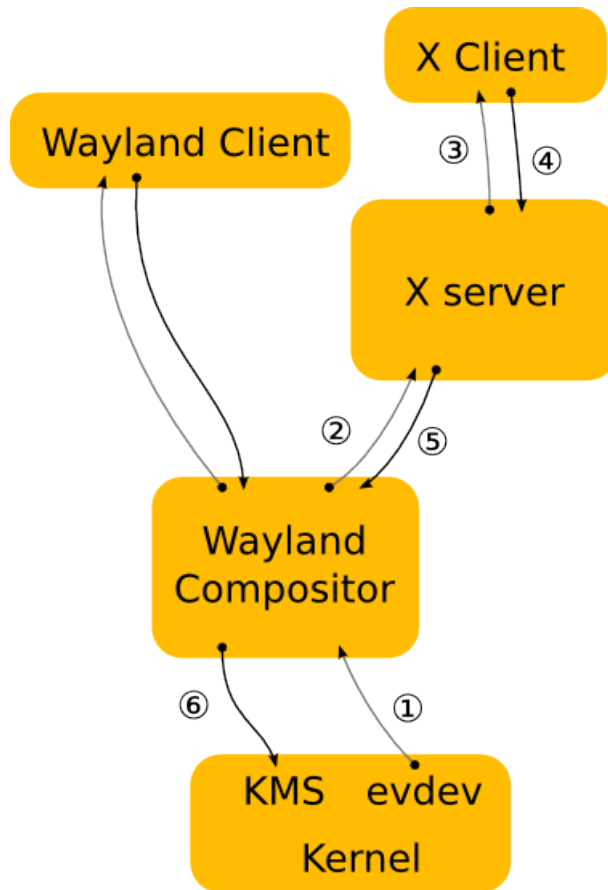


Figure 4.8: X on Wayland

4.4.3 Ubuntu Mir - Deprecated

Not to be outdone Ubuntu saw an opportunity to come up with an X replacement and announced the Mir project in early 2013. Unfortunately this has met with serious opposition from the X and Wayland communities seeing this as an outside effort to “corner the market”, even to the point in 2013 of Intel, who supports Wayland, [rejecting X Mir support](#) in its Linux opensource graphic driver package.

Ubuntu is the only Linux operating system to support Mir and development has been delayed as the project has taken on the monumental task of replacing X. Ubuntu was criticized for not joining the Wayland group’s work but I think that may have been somewhat without warrant. Ubuntu has a business use case—they are looking to make a compositor that could adapt based on form factor, something not unlike how Android works. This would enable them to make Ubuntu for tablets, phones, TVs, even smaller devices electronic devices. Mir would be customizable to Ubuntu’s hardware dreams and they could sell hardware and perhaps make some money.

*“In March 2014, Mark Shuttleworth confirmed that Mir development had been delayed and that it was now forecast to be default for desktop use in Ubuntu 16.04 LTS, expected to be released in April 2016.”*³

As of [April 5th 2017](#), Mark Shuttleworth announced that in Ubuntu 18.04 the UNITY interface and the Mir compositing engine would be deprecated in favor of GNOME 3 and Wayland. But the Mir Project and Unity lives on in the Ubuntu Touch port maintained for phones and tablets by the [Ubuntu-Touch.io](#) project. The Mir display server is now gaining popularity running Ubuntu Touch on the [Librem 5](#) and the [Pinephone](#) Linux based cell phones.

³<http://www.omgubuntu.co.uk/2014/03/mir-default-display-server-ubuntu-2016>

4.5 Window Managers

X will let you display a window and let you capture mouse and keyboard commands. But what if you want to render multiple windows on one machine and manage their state as well such as opening and closing them. How about arranging them? Do you need them tiled or overlapping? Then you need a window manager to sit on top of X and handle that rendering. Thankfully window managers are essentially common in Desktop Linux and almost inseparable from X. Window Managers are not the full GUIs you might be used to. They are one step below a full blown desktop environment but window managers are very fast because they render simple windows and simple sets of icons. Their job is to render content inside of particular windows and manage that window's state. Since Linux and X were developed independently, you will see a depth and breadth of different window managers serving different functions and features. Some are very simple, some serve direct purposes, some are tied to full blown desktop environments. You can break window managers into 3 categories based on their window behavior.

4.5.1 Compositing

The first compositing window managers came about in 2005/2006 as the necessary extensions were added to the X server. The compositing window managers are now considered standard and default part of desktop environments such as GNOME, KDE, and XFCE.

- [Xfwm](#) - Xfce window manager
- [KWin](#) - Window manager for the KDE Desktop Environment
- [Mutter](#) - Window manager for the GNOME3 Desktop Environment

4.5.2 Stacking

These allow for overlapping windows, like Windows or MacOS. The LXQT desktop environment uses [Openbox](#) as it's window manager.

- [Afterstep](#) - NEXTSTEP look and feel
- [Blackbox](#) - Similar to the NeXT interface and Window Maker
- [Fluxbox](#) - Highly configurable and low resource
- [FLWM](#) Fast and Light Window Manager
- [FVWM](#) - Minimize memory consumption, provide a 3D look to windows, and a virtual desktop
- [IceWM](#) - Win95-OS/2-Motif-like window manager
- [Openbox](#) - Standards compliant, fast, light-weight, extensible window manager

4.5.3 Tiling

Tiling Window Managers place window next to each other like tiles on the floor and are non-overlapping. This style is popular with many coders and developers who want to see many windows open at once.

- [dwm](#) - Dynamic window manager
 - [dwm tutorial](#)
- [i3](#) - Good documentation, reasonable defaults, and good multi-monitor support
 - [i3 keyboard command reference](#)
- [Xmonad](#) - Minimalist, tiling window manager written in Haskell
 - [Xmonad User Guide](#)

4.6 Desktop Environments

About the time that the OSI movement was launching in late 1990s, there was a growing need for more than just window managers too. x86 Intel and AMD CPUs at that time were gaining in processing power and processor based multimedia extensions began to appear by 1997, collectively called MMX. What is the difference between a desktop environment and a window manager? Simple things that you may take for granted such as a clock, or a text editor, office suite or an email client, even a web browser. Also a desktop environment provides an IPC method for inter-window communication. Most important, a desktop environment includes a file manager application, a start or

action menu feature and a login manager (display manager) such as GDM, KDM, or SDDM⁴. On top of that all the elements of the desktop have a changeable but consistent usage pattern and look-and-feel. Let us look first at the development history of the two main desktop environments followed by a few other desktop environments.

4.6.1 KDE⁵

The KDE project (originally the K Desktop Environment) was the first opensource Linux desktop environment project. Started by Matthias Ettrich at the university of Tübingen in Germany in 1996. The first release of the **K Desktop Environment** was in 1998 and the name was a “*clever hack*” of the CDE–Common Desktop Environment–developed for Unix by Sun, HP, and IBM. KDE focused on the lack of perceived usability in Linux window managers. You and I may take desktop environments for granted based on our experience with Mac and Windows respectively. Unix and Linux in 1998, did not have the same capabilities–just a mix of window managers and independent applications. At the time of development there were only a few toolkits available that could be used for creating desktop environments in Linux. One of them was Qt, pronounced “*cute-ee*”. Qt was a good choice for developers because at the time was the only toolkit that had C++ language bindings. GTK was C only at this point. Matthias Ettrich chose the best technology at the time for developing GUIs. Qt was initially developed by a company called Trolltech.

In 1998 Qt was not available under a completely “*free software*” license. It had a modified license called the QPL, stating that non-commercial software projects for Unix/Linux were allowable. By 2000 Trolltech relicensed Qt to be GPL compliant. This fact still upset Richard Stallman who never forgave KDE for initially using a non-GPL compatible license. As an aside, Nokia bought Trolltech in 2008 but Qt remained under GPL license.⁶ Qt was used by Nokia to power their Symbian OS which dominated the cellular market as the predominant OS until Android unseated it by 2010.

KDE is unique because although Qt has vendor input to the project, KDE itself is not vendor owned. KDE is also not tied to a single Linux distro in the way that GNOME and Red Hat are intricately tied together now. Distros such as [openSUSE](#), [Mageia](#), [Chakra](#), [Kubuntu](#), [PCLinuxOS](#) and even FreeBSD are using KDE by default or have it available in software repositories. Qt recently split itself during the 5.0 release from one entire library into [three separate sub-components](#):

- KDE Desktop - Desktop called [Plasma 5](#) released every three months
- KDE Applications - [KDE applications](#) updated and released every four months
- KDE Frameworks - [Common file, os, and services](#)

4.6.2 GNOME⁷

Shortly after the initial release of KDE and its licensing confusion, an enterprising opensource advocate saw the need for a truly open and free desktop environment. His name is [Miguel De Icaza](#)

4.6.2.1 Miguel de Icaza

GNOME initially stood for GNU Network Object Model Environment but the acronym is no longer used⁸. GNOME was a GNU project directly created under the GPL. The first major GNOME release was done by Miguel in March of 1999. This project used the [GTK+](#), commonly mistaken as the *Gnome Tool Kit*, actually stands for the GIMP Tool Kit. [GIMP](#) is the GNU Image Manipulation Program similar to Photoshop or Paint.Net in feature set. GTK or now known as GTK+ is a library useful for creating standalone applications with multiple development language bindings as well.

In an interesting development Miguel ended up forming the company that became Xamarin, a cross development mobile platform company using C# to develop for Android and iOS–shortly after Microsoft began opensourcing the C# language. Recently Xamarin was purchased by Microsoft and Miguel is now a Microsoft VP. Miguel has always been a software pragmatist - always chasing the best technology and open sourcing it. He took much criticism because he started the Mono project in bringing Microsoft’s C# and .NET platform to Linux, because he saw C# as the best language to develop in cross platform. This brought about the condemnation of Richard Stallman, calling

⁴KDE Plasma 5 retired KDM as the default display manager in favor of SDDM.

http://www.phoronix.com/scan.php?page=news_item&px=MTgyOTU

⁵Comparison of X Windows Desktop Environments https://en.wikipedia.org/wiki/Comparison_of_X_Window_System_desktop_environments

⁶<http://arstechnica.com/information-technology/2008/01/nokia-buys-trolltech-will-become-a-patron-of-kde/>

⁷Comparison of X Windows Desktop Environments https://en.wikipedia.org/wiki/Comparison_of_X_Window_System_desktop_environments

⁸Miguel de Icaza Licensed under CC BY 2.0 via Commons.



Figure 4.9: *Miguel de Icaza*

Miguel, “*A traitor to free software.*”⁹ Miguel’s response to Stallman was a beautiful example of how to diffuse an inflammatory situation:

*“I want to say that God loves all creatures. From the formidable elephant to the tiniest ant. And that includes Richard Stallman.

As for me, I think that there is a world of possibility, and if Richard wants to discuss how we can improve the pool of open source/free software in the world he has my email address.

Love, Miguel.”*¹⁰

In the early 2000s Sun and HP adopted the GNOME desktop as the replacement for CDE for their Unix distros. Red Hat and Debian adopted GNOME as well. With the release of Ubuntu 18.04, Ubuntu returned to the Gnome desktop.

GNOME 2 was released in early 2002, GNOME 3 was released in 2010, and GNOME 4—now known as GNOME 40, released March 2021. Over this 18+ years GNOME developed itself as a mature desktop by using the traditional desktop metaphor: start menu, task managers, and toolbars. But like all things that change, in early 2011 GNOME 3 was released, and there was a mighty backlash. Even Linus Torvalds, who is a Red Hat and GNOME user came out against GNOME 3 very hard, calling it a “*total UX disaster*”¹¹. What made GNOME 3 so different is that it took on a different metaphor than say Windows OS, called the GNOME shell. Think of the GNOME shell as a way to manage or view multiple tasks and applications happening at once. Instead of a single desktop, think of the new metaphor as a table top using the shell as a magnifying glass to view all tasks. The GNOME naming convention has changed from the previous 2.x and 3.x to just 40. So there is in a sense no more GNOME 3 or 4, but like a webbrowser just a rolling version starting at the number 40.

4.6.2.2 GNOME Forks: Unity, Mate, Cinnamon, and Pop!_OS

Linus Torvalds was quoted as saying:

“People don’t want Gnome 2 because it was a pinnacle of some GUI design. They want it because Gnome 3 removed features they used, and made it harder to get their work done,”¹²

GNOME 3 led to the creation of alternatives in 2012. When GNOME moved from version 2 to version 3 the amount change was seen by some GNOME users as treason. The MATE project (pronounced “*ma-tay*” not “*may-tuh*”) was a fork and continuation of the GNOME 2 code base. The Cinnamon desktop was a fork of GNOME 3 with an added features set for the Linux Mint distro. Unity was Ubuntu’s reimplementaion of the GNOME shell with the Unity shell on top of GNOME 3. These desktops, with the exception of Unity, can be deployed or installed on most Linux distributions. As a final thought Linus Torvalds has since reconciled with GNOME 3 as of 2013 based on some additional 3rd party tools that allow the GNOME 3 experience to be customized to his liking.¹³

⁹http://www.osnews.com/story/22225/RMS_De_Icaza_Traitor_to_Free_Software_Community/

¹⁰Miguel’s response to Stallman’s accusation <http://tirania.org/blog/archive/2009/Sep-23.html>

¹¹<http://www.zdnet.com/article/linus-torvalds-finds-gnome-3-4-to-be-a-total-user-experience-design-failure/#!>

¹²<https://plus.google.com/115250422803614415116/posts/KygiWsQc4Wm>

¹³<http://www.zdnet.com/article/linus-torvalds-switches-back-to-gnome-3-x-desktop/>

In April of 2021, [Linux open laptop manufacturer, System76](https://pop.system76.com/) released a themed GNOME 3 desktop called [Pop!_OS](https://pop.system76.com/ “Website to Pop OS”) with a custom windows manager that defaults to a tiling mode rather than a stacking mode.

GNOME is considered the Linux default desktop due to its tight integration with systemd. So much so that GNOME community contributed development has begun to wane ¹⁴. Which is better GNOME or KDE? With all major Linux systems that are using systemd you now have no choice but to adopt GNOME as the default desktop. Both have had great set backs and great feature advancements over the years. In the end it is up to you based on your usage patterns, development environment preferences, even which distro you use or prefer. Try them both and choose the best one for your needs.

4.6.3 Xfce

Seeing as KDE and GNOME focused on features and usability, many people who were using older hardware felt left out or unable to run these environments as the resources required were growing. So a movement to create a lightweight desktop environment sprung up. The first was Xfce and was developed in parallel to KDE and GNOME by Olivier Fourdan. Xfce wrote all of its [components](#) by itself not relying on or forking any of GNOME. This environment had a focus on simplicity and running with lower end hardware requirements. Originally based on a proprietary toolkit, when Xfce was rejected for inclusion in Red Hat Linux because of this license, Xfce moved to GTK+ 2 toolkit by 1999. At one point Debian was considering it as an alternative to GNOME, but dropped it because of adopting systemd—forcing Debian to take GNOME 3 as their default desktop. Xubuntu is a common derivative distro made up of Ubuntu using Xfce instead of GNOME. Xfce until recently was using the GTK2 toolkit but is in the process of being rewritten in GTK3 conversion started with version 4.12 in 2015. As of 2018, most of the major components have been re-written in GTK3. When version 4.14 is released all major components will be converted to GTK3.

4.6.4 LXQT

The LXQT project was started as the LXDE project in 2006 by Hong Jen Yee. This desktop environment is even more spartan than Xfce but is one step above a window manager. LXDE’s focus is on making pretty much any laptop or PC made in the last decade still usable for modern Linux. Recently the lead developer Hong Jen Yee had disagreements with the direction GTK+ 3 was taking and has made a parallel design port called LXQT. LXQT involved LXDE porting their desktop applications to Qt and merging with a defunct project called Razor-Qt to produce LXQT.

4.6.5 Enlightenment and Lumina

This project started in 1997 and intended to be a virtual desktop window manager. This would involve a grid of desktops a user could move windows around and onto. The initial release was in 1999 called E16. The second release E17 took 12 years. But E17 had grown from just a window manager into a full-fledged desktop environment using its own libraries called Enlightenment Foundation Libraries. Enlightenment can also be a shell that can sit on top of KDE and GNOME.

Lumina is a desktop environment created for a distro of FreeBSD, called TrueOS. They had previously relied on GNOME and KDE but the amount of work needed to remove the Linux specific parts prevented new research from being done. As of 2019, TrueOS rebased itself to be part of Project Trident and based on Void Linux and became a Linux distro but continues to use the Lumina desktop.

4.6.6 Android

Android runs on the Linux Kernel and is in a sense a custom Linux Distro itself. [It has a custom rendering layer](#) that does not use X or Wayland.

4.6.7 Who Uses What

KDE 5 ¹⁵	Qt 5	https://www.kde.org/
GNOME 40	GTK+ 4	https://GNome.org/

¹⁴<https://blogs.gnome.org/otte/2012/07/27/staring-into-the-abyss/>

Xfce	GTK+ 3	https://www.xfce.org/
LXQT	Qt 5	https://lxqt.org/about/
MATE	GTK+ 3	https://mate-desktop.org/
Cinnamon	GTK+ 3	https://cinnamon.linuxmint.com/
Unity	GTK+ 3	https://unity8.io/
Enlightenment	EFL	https://www.enlightenment.org/
Lumina	C++/Qt5	https://lumina-desktop.org/

Wikipedia has a sample gallery of all these desktop environments and more at https://en.wikipedia.org/wiki/Desktop_environment#Gallery. As always there are many other desktops we couldn't cover. Also keep in mind that most of these have been ported to work on the various BSDs as well.

4.6.8 Gnome 3.3.x Features

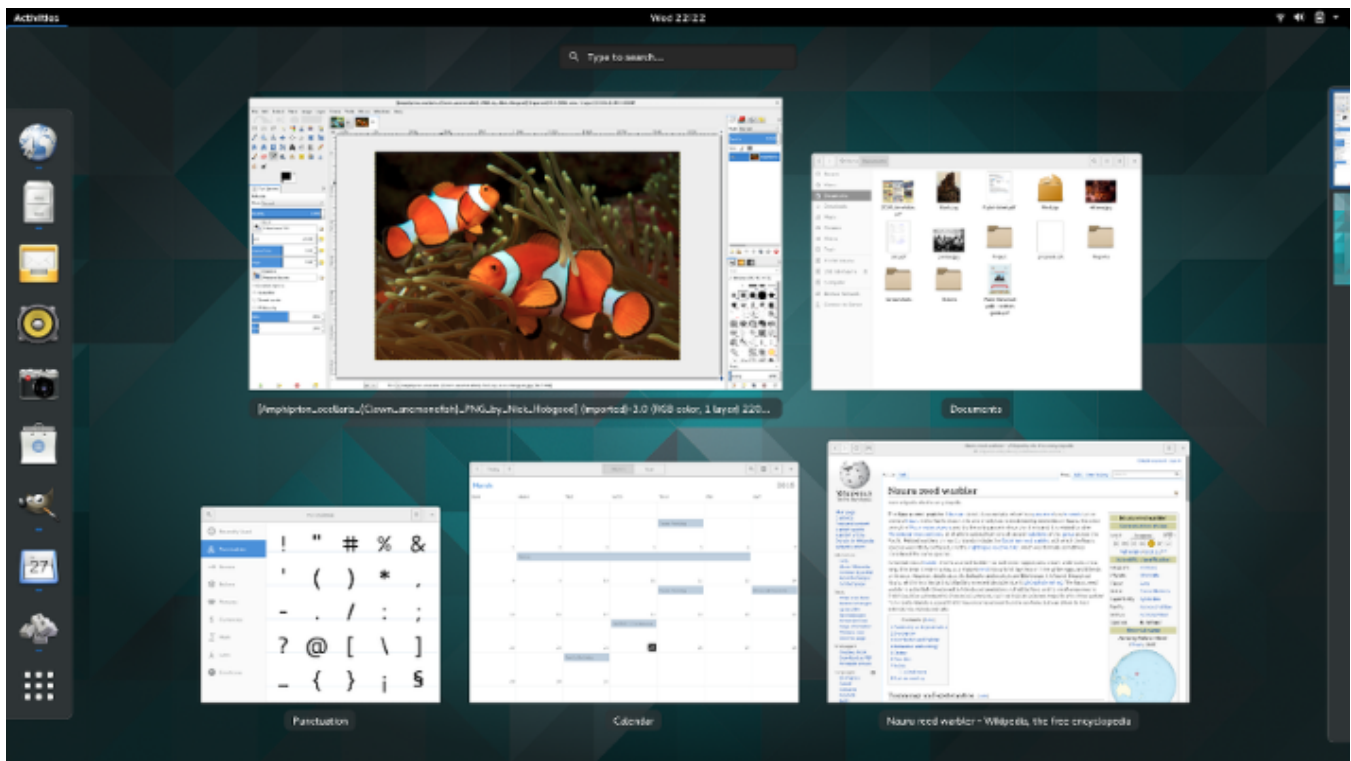
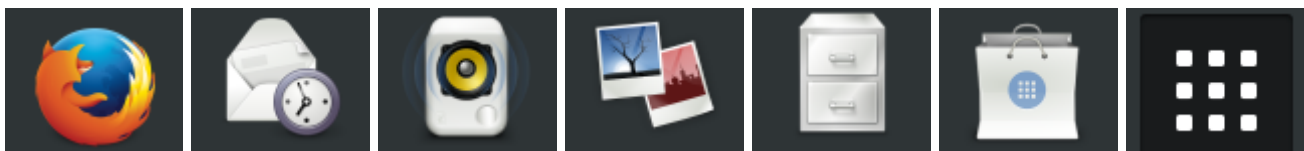


Figure 4.10: GNOME 3

Everything in **GNOME 3** starts with the **Activities** button in the upper left hand corner. There is no traditional “Start Button.” Just this sense of activities to be launched. There is also a **Find** box for searching everything in GNOME 3 from documents to applications to even suggested software. There is also a **favorites bar** on the left hand side of the screen. You will see preinstalled icons for Firefox Web Browser, Evolution E-mail client, Rhythm Box music player, Shotwell for picture management, the file manager is named Nautilus, the Software Store application, and finally a grid launcher for everything else. Note there are no “categories” such as system tools or office software. This is a design feature of GNOME 3.



¹⁵Qt 5 / KDE 5 split into three separate components https://en.wikipedia.org/wiki/Qt_%28software%29#Qt_5

You will notice a few other system designs or features. For instance there is no logout button. The little white triangle in the upper right hand corner next to the battery and sound icon controls poweroff and restart, but by default there is no logout button. This is a GNOME feature.

Each application has a preferences and content menu area in GNOME 3. It is located right next to the Activities button. You will see the application name with a small white drop down arrow. In order to find the window commands for open, close, resize, tiling, and so forth you just need to right click on the gray toolbar on top of the window that has the focus.

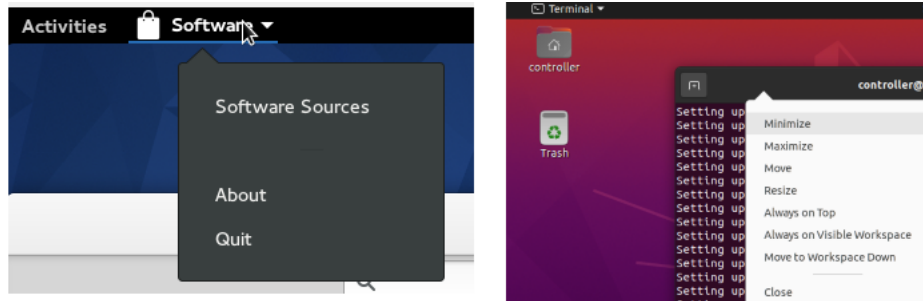


Figure 4.11: *GNOME 3 Taskbar*

4.6.8.1 GNOME Remove and Add Favorites

You can remove default icons from the favorite menu as well as add your own favorite applications in GNOME 3 and GNOME 40. By right-clicking on the icon you have an option to remove it from the Favorites menu.

For example to add a Favorite for the Terminal program click **Activities** > **Search** > type **Terminal** > right click on the icon > **Add to Favorites**.

4.6.9 GNOME 40 Features

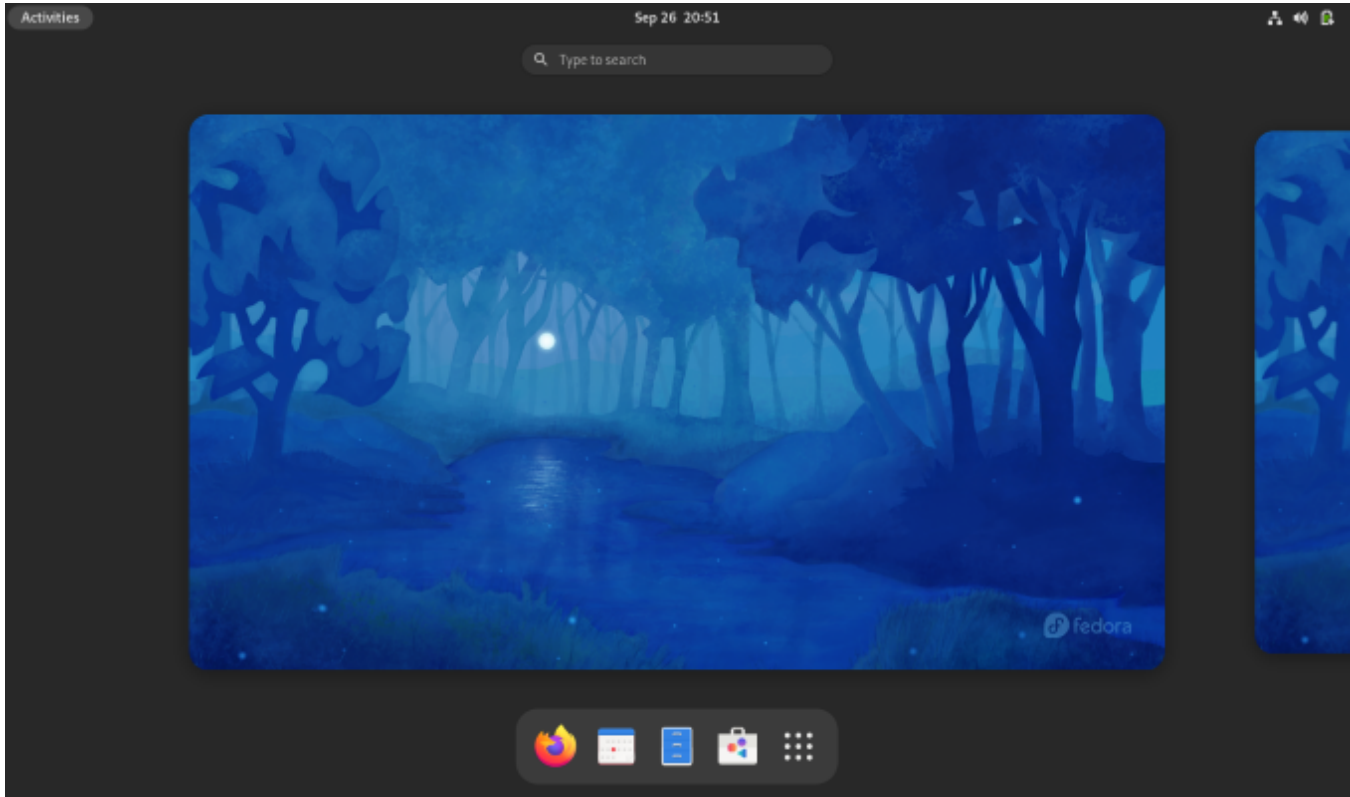


Figure 4.12: GNOME 40

The major change in GNOME 40, which as of September 2021 only Fedora 34 has adopted, is that the application doc is now located along the bottom of the screen and no longer vertical. There have been numerous internal app changes as well as cloud service integrations. There have been changes to active application on the doc.

There has also been an integration of track pad gestures. “Even if you’re not a laptop user, you can still use these gestures by using the keyboard shortcut: Super+Alt+↑ or ↓ or ← or →. Going up and down on the touchpad takes you in and out of the activities overview and app grid screens, left and right switches you between workspaces. And, if you’re using a mouse, you can navigate using Super + Alt + Scroll.”

4.6.9.1 GNOME Extensions

To extend the functionality of GNOME there is a [GNOME extensions website](#). This site allows you to add on additional features to GNOME that you may want. If visit the site while using Fedora or any GNOME enabled site and you allow the GNOME plugin to run in your Firefox browser, then the website will allow you to interactively install plugins. Try it. Go to the second page and choose a plugin called *Caffeine*. This is handy because it allows you to disable the screensaver and system suspend and is very handy if you are working on a virtual machine. Note how the caffeine plugin simply places a coffee cup icon in the top toolbar? A few others you may want to try are the [Drop Down Terminal](#), [Logout Button on Panel](#), or the [System Monitor](#). Most GNOME 3 extensions should work on GNOME 40 but not all have been ported and some are in the proccess as GNOME 40 is still relatively new.

4.6.10 Installing Windows Managers and Desktop Environments

Previously Linux distributions had made an effort to build in Software Stores, much like we saw in iOS and Android. The issue with a software store became the ability to make money, charge money, and distribute money. All though the distributions still support Software Stores, their heart is not in this manner of software installation. The original way to install software was via a package manager, which the stores were a front for anyway. There are two competing standards to replace packages or more appropriately bring “app” like functionality to Linux. These are called [Flatpak](#) on Fedora and [Snaps](#).

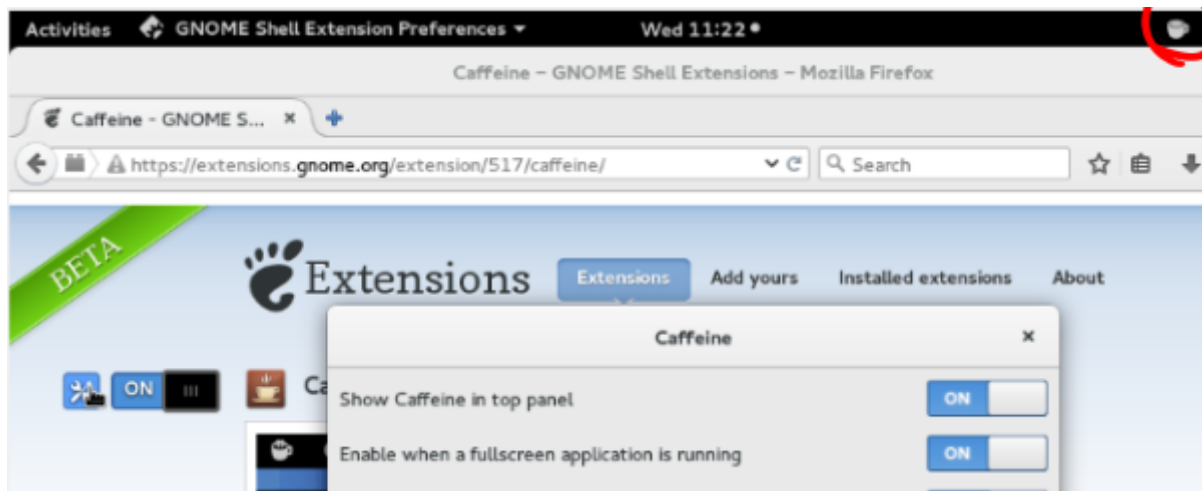


Figure 4.13: *GNOME Extensions*

To access the terminal on a GNOME desktop, click the **Activities** button at the top and in the **find** box type: *terminal*.

4.6.10.1 Ubuntu

```
sudo apt-get update    then    sudo apt-get install i3 icewm fvwm flwm evilwm fluxbox dwm
xmonad mwm xfce4
```

4.6.10.2 Fedora

```
sudo dnf install i3 icewm xmonad mwm @xfce
```

4.6.10.3 GNOME Software Store

This is a feature added by Red Hat to improve upon Gnome Packages, which is the default package manager in GNOME 3. The package manager is called **Software**. There is a wide selection of free and opensource software to choose from. In the meantime you can use the Software store to install the good old **GNOME Packages** and **GNOME Update** package manager and updater which will default to the old style and let you install pretty much everything. The Software store fits a niche need along side the Snap and Flatpak stores, as the GNOME software store focuses on software that integrates with the Desktop, and not so much on standalone applications that the other package managers focus on.

4.7 Conclusion

In this chapter we learned about the creation and evolution of the desktop GUI. From ttys to X to Wayland and a modern compositor. We also learned about window managers and the four major desktop environments. Finally we learned about the major GUI desktop environment development kits and how each major DE uses them on the Linux GUI desktop.

4.7.1 Review Questions

- 1) What was the original and most popular Unix “Smart Terminal?”
 - a. VIC-100
 - b. VT-100
 - c. VT-220
 - d. VC-100
- 2) What is the three letter abbreviation still in use today in modern Linux to refer to “terminal devices?”

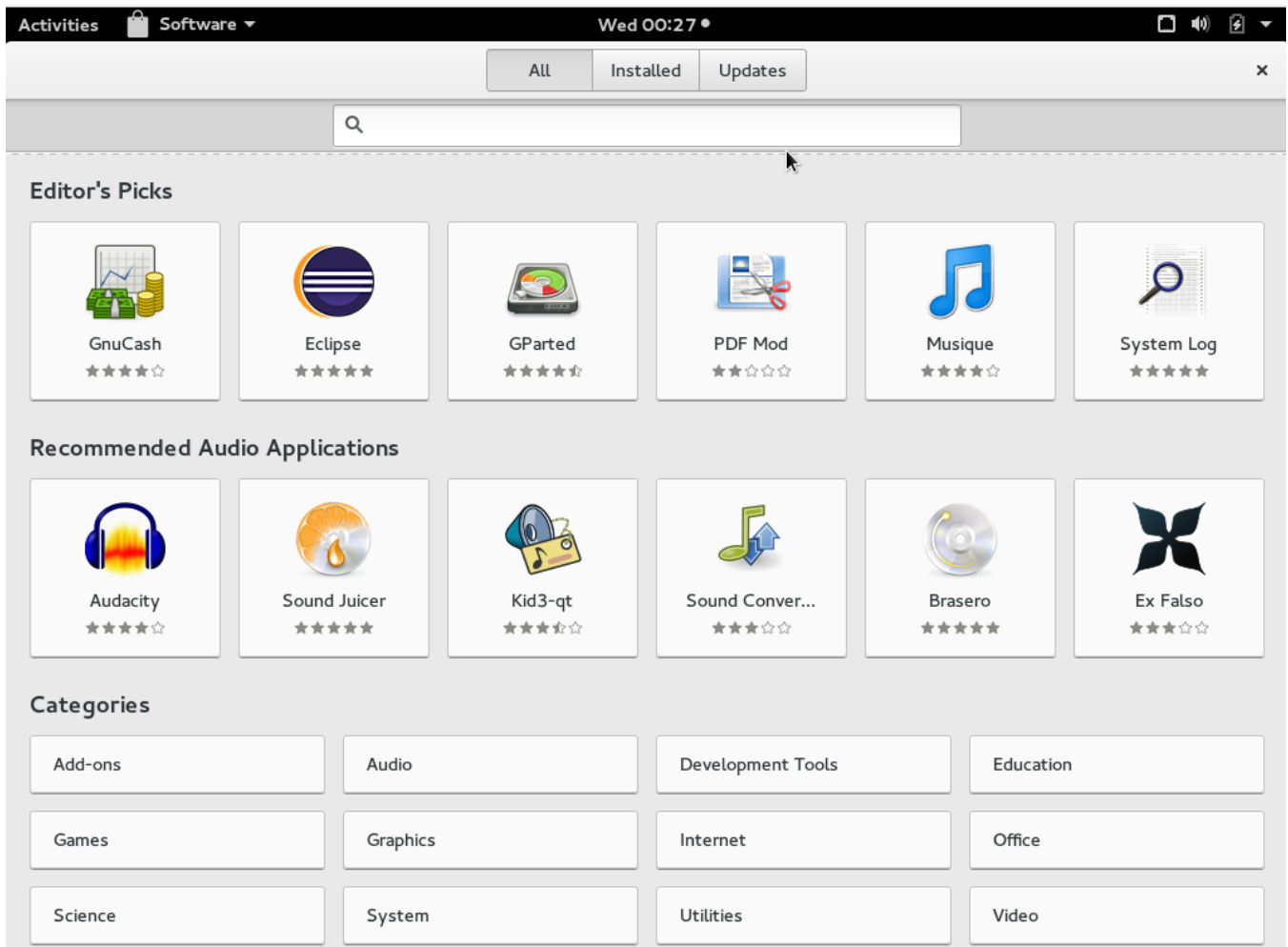


Figure 4.14: GNOME Software Store

- a. TCY
 - b. TTY
 - c. VT-100
 - d. Virtual Terminal
- 3) Why is the GNOME terminal and Windows cmd.exe terminal emulator screens 80 by 25 by default?
- a. The technology cannot process any larger size
 - b. The technology doesn't need to have any larger screen size
 - c. The developers of these technologies were seeking to emulate the popular VT-100 and VT-220 terminals they used prior.
 - d. Hey it's Windows, do they need a reason?
- 4) What is the key combo sequence you can hit to switch to a new virtual terminal in Linux?
- a. Alt + Ctrl + Del
 - b. Alt + Ctrl + F1 - F7
 - c. Right Ctrl
 - d. Shift 5 times
- 5) What is the name of the original Unix based GUI that came out of MIT in ~1984?
- a. W
 - b. X
 - c. Y
 - d. Z
- 6) True or False – X was originally not opensourced by MIT in 1984
- 7) What is the model that the X server uses to render screens?
- 8) What is the name of the successor GUI compositor to X being created by the X.org foundation?
- a. KDE
 - b. Y
 - c. Wayland Project
 - d. Jennings Project
- 9) What is the name of Ubuntu's, now deprecated, GUI compositor replacement for X?
- 10) What are the 3 types of Linux window managers?
- 11) There are 4 major Linux desktop environments in use today: which grouping is correct?
- a. KDE, GNOME, CDE, LXDE
 - b. KDE, GNOME, E17, GNUMSTEP
 - c. KDE, GNOME, X, LXDE
 - d. KDE, GNOME, Xfce, LXDE
- 12) What is the name of the windowing toolkit that KDE uses?
- 13) What is the name of the windowing toolkit that GNOME uses?
- 14) What is the name of the founder of the GNOME project?
- 15) When the GNOME 3 desktop environment was released in early 2012, many people were unhappy that many changes were made. There were 3 major projects started to either preserve GNOME 2 or to modify GNOME 3 significantly – what are the names of those projects?
- a. Enlightenment, LXDE, Xfce
 - b. Mint, Unity, Mate
 - c. Unity, Mate, Cinnamon
 - d. Cinnamon, Mint, Mate
- 16) When released in 2006, what was the main development goal of the LXDE desktop environment?
- a. GUI desktop features

- b. Multi-desktop paradigm
 - c. Energy saving and extremely fast
 - d. Made for high end gaming systems
- 17) Which of these statements are true in regards to Linux desktop environments? (choose all that apply)
- a. Desktop environments have a File Manager
 - b. Desktop Environments have start or action buttons and task and notification bars
 - c. Desktop environments have a changeable but consistent look-and-feel
 - d. Desktop environments have system configuration tools and user applications built in
 - e. Desktop environments have lower memory requirements than window managers
- 18) What is the default desktop environment and version for the latest Fedora desktop (Name and version)?
- 19) What is the name of the button on the upper left hand corner of the GNOME desktop that you use to “launch applications?”
- 20) What is the name of the desktop environment used in Xubuntu?

4.7.2 Podcast Questions

Please answer these questions from the Fedora Project podcast on [FLOSS](http://twit.tv/floss/71) - <http://twit.tv/floss/71>

- ~9:11 What is the Fedora Project?
- ~11:35 How does Red Hat make money on Fedora if it is free?
- ~12:30 What is the Fedora release cycle and can businesses use this release cycle?
- ~13:30 What is the relationship between Fedora and Red Hat Enterprise Linux (RHEL)?
- ~25:00 What percentage of the Fedora Project is open source?
- ~35:00 On further inspection is Fedora Project really opened source according to the Free Software Foundation?
- ~36:20 Does Fedora include proprietary Nvidia drivers? Why or why not?
- ~44:30 Who is the most famous Fedora user?
- ~1:01:00 What is the difference between CentOS and RHEL?

4.7.3 Lab

Using the virtual machines you installed in the previous chapter, Fedora 34 and Ubuntu Desktop 20.04, you will now install additional software, themes, desktop environments, and window managers. You will combine screenshots of this newly installed software into a single document for submission. We will be focusing on package based installs in the first part of the lab.

4.7.3.1 GNOME 3 Extensions

You will need to install the listed GNOME 3 extensions from <https://extensions.gnome.org>. The first one is the example in the book called *Caffeine*. Make sure you have the Gnome Shell integration installed to configure and install plugins directly from the browser.

- 1) Caffeine (screenshot of coffee cup icon in upper right corner)
- 2) Dash to Dock (move dock to the bottom of the screen)
- 3) Screenshot Tool (screenshot of camera icon in upper right corner)

4.7.3.2 Snaps Install

To install and configure snaps, reference chapter 3. To show these packages are installed take a screenshot of the output of the command: `sudo snap list`. Install these packages via Snap on any **Ubuntu or Debian** based desktop:

- 1) Android Studio
- 2) Blender
- 3) Slack
- 4) Discord
- 5) Visual Studio Code

4.7.3.3 Flatpak Install

To install and configure flatpak and flathub, reference chapter 3. To show these packages are installed take a screenshot of the output of the command: `sudo flatpak list`. Install these packages via flatpak on the **Fedora** virtual machine:

- 1) Kdenlive
- 2) Visual Studio Code
- 3) Discord
- 4) Remmina
- 5) GtkStressTesting

4.7.3.4 Ubuntu 20.04 Flatpak Install

To install and configure flatpak and flathub, reference chapter 3. To show these packages are installed take a screenshot of the output of the command: `sudo flatpak list`. Install these packages via flatpak:

- 1) Microsoft Teams
- 2) Shotcut
- 3) LibrePCB
- 4) OpenBoard

4.7.3.5 AppImage Install

Find 3 [AppImage install packages](#), follow the instructions to install and run these AppImages on both Ubuntu 20.04 and Fedora 34. Take a screenshot of the command needed to run the AppImage and the corresponding first screen of that application from the [AppImage GitHub repo](#).

- 1) Poddr
- 2) Archipelago
- 3) LibrePCB

4.7.3.6 Installing Window Managers

You will chose 1 of the window managers from the categories listed earlier in the chapter and install them. Once installed you need to log out and restart your session. In order to change the default window manager on GNOME3 desktop environment you need to click on the user name and select the little gear below the password field. In your screenshot open a terminal window, you might need to do some research to understand how to operate in a window manager.

- Stacking window manager - [Openbox](#)
 - Show 3 Windows stacked (FireFox, Terminal, and the File Manager)
- Tiling window manager - [i3](#)
 - [i3 keyboard command reference](#)
 - Show 3 Windows tiled, 2 Terminal Windows and 1 FireFox.
 - Use the link above to learn the key commands to launch windows, remember there is no mouse!

Note: the names of packages are not always obvious so you can use search features of package managers. For example, here would be how to search for the i3 package.

- `sudo apt-cache search i3`
- `sudo dnf search i3`

4.7.3.7 Desktop Environment Installs

Install these desktops, restart your system and as you login switch your desktop environment and take a screenshot of the new environment.

- 1) Install the Xfce Desktop on Fedora 34
- 2) Install the Ubuntu Mate Desktop on Ubuntu 20.04
- 3) Install the Xfce4 Desktop on Ubuntu 20.04 (the xubuntu-desktop package)

Deliverable: Take screenshots of all successful installs of software. See Blackboard for submission details.

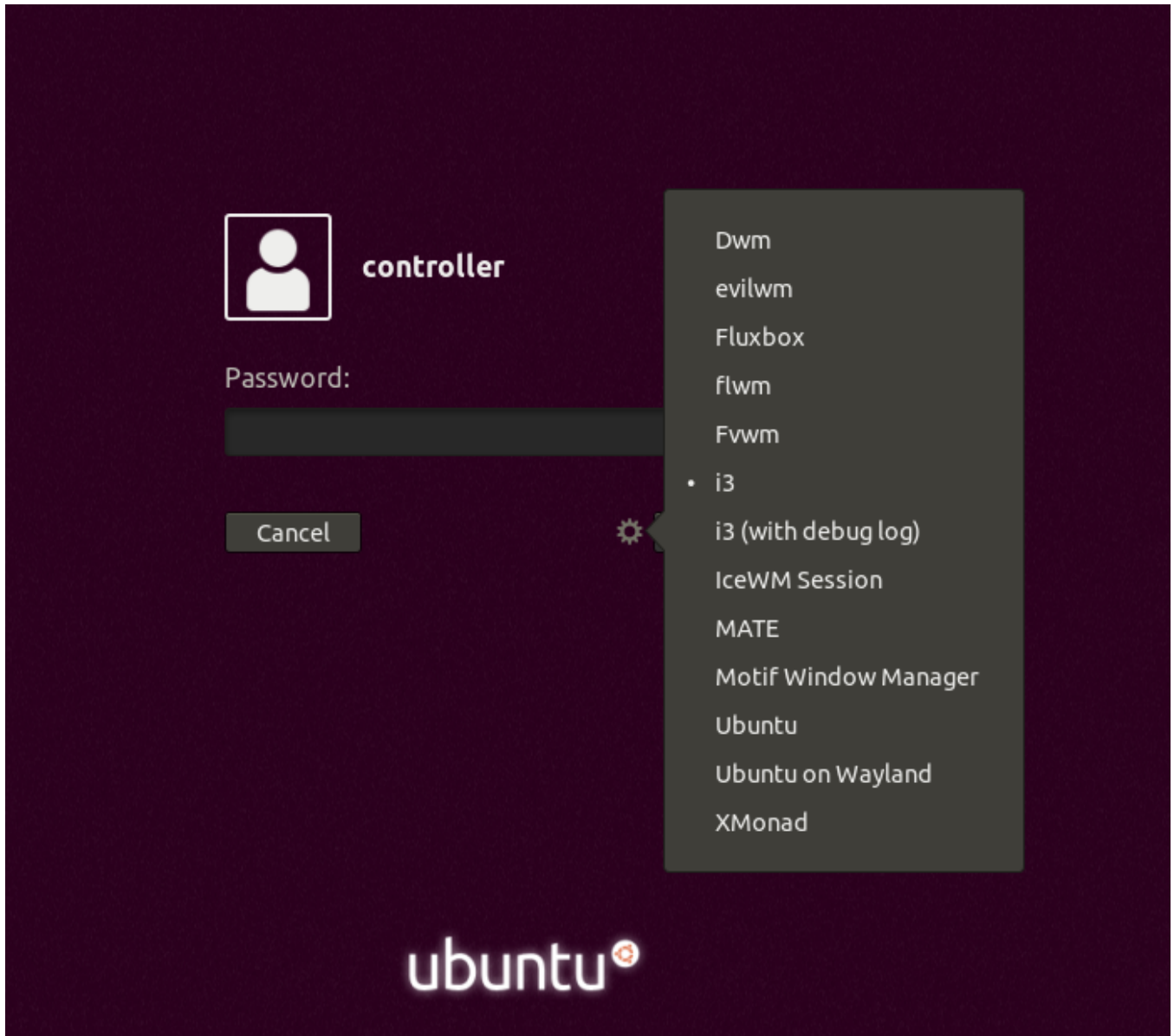


Figure 4.15: *Change Window Managers*

4.7.3.8 Footnotes

Chapter 5

The Linux Filesystem, Path, Shell, and File Permissions

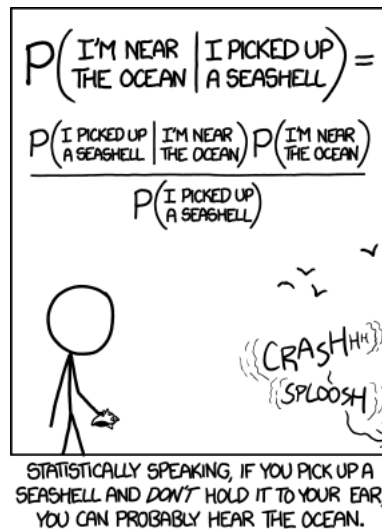


Figure 5.1: *The Shells*

5.1 Objectives

- Understand the structure of the Linux Filesystem
- Understand the difference between absolute and relative paths
- Understand the function of the Linux Shell and its relation to the operating system
- Understand the fundamental shell commands for traversing and modifying the contents of the filesystem
- Learn the Linux commandline nomenclature
- Understand the nomenclature describing file permissions and security

5.2 Outcomes

At the completion of this chapter you will understand how to use the Linux shell for modifying the contents of the operating system. You will understand the nature of the filesystem and how to navigate it. You will be able to demonstrate full awareness of absolute and relative paths and understand the system path. You will know basic shell commands for manipulating content in the filesystem.

5.2.1 Chapter Conventions

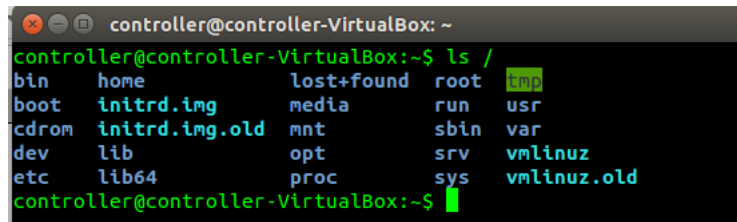
The terms **commandline**, **terminal**, and **shell** have been used interchangeably so far to convey the method for a user to issue commands to the kernel. In this chapter we will explore the discrete differences between the terminal and the shell, with the term commandline or CLI being a generic word referring to anything where system commands are entered in text.

The terms **files**, **folders**, and **directories** can be used interchangeably in regards to this chapter’s contents. Remember from Chapter 2, everything in Unix is a *file*, meaning that a folder which is also a directory is also a file from the operating systems point of view. You will also notice a tag **Exercise:** in blockquotes throughout the chapter with little exercises that will let you follow along as you read.

5.3 What is a Filesystem

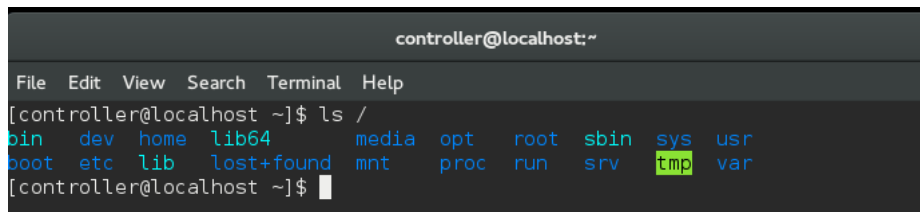
A filesystem is a way for the operating system to manage and access files. It is how data is segregated and the mechanism the operating system uses to retrieve and write data. The benefit of a filesystem is it will keep track of the locations in memory of your files. When you double click on a jpg picture in your photo directory, the operating system talks to the filesystem and says, “I want this picture, where is it located?” The filesystem has an index table of all files location in memory and looks up the address of the file. That address is given to the kernel which then passes it to the disk controller for the data retrieval portion. The bits are effectively returned to the operating system from the kernel and the image is rendered to the screen.

The Linux filesystem was designed as an **upside down tree** by Ken Thompson. What is at the bottom of a tree? The **root** is at the bottom. This is an important concept to remember in Linux. In Linux the **root** or written as “/” (pronounced *slash*), is at the top of our upside down tree. From the **root** you navigate down the file hierarchy using the **cd** commandline tool. You cannot go any higher in the tree than **root**, you can only go down. When you type the command **ls /** you see a directory listing of the contents of the root directory. As you can see below the root directory contains sub-folders, and those sub-folders can have many more sub-folders. This first level below the **root** has a common core of directories across each Linux distro. Let us take a closer look at Ubuntu and Fedora root folder file hierarchy.



```
controller@controller-VirtualBox: ~  
controller@controller-VirtualBox:~$ ls /  
bin      home      lost+found  root      tmp  
boot    initrd.img  media      run       usr  
cdrom   initrd.img.old  mnt       sbin     var  
dev     lib        opt        srv       vmlinuz  
etc     lib64     proc       sys       vmlinuz.old  
controller@controller-VirtualBox:~$
```

Figure 5.2: *Ubuntu root directory listing*



```
controller@localhost:~  
File Edit View Search Terminal Help  
[controller@localhost ~]$ ls /  
bin  dev  home  lib64  media  opt  root  sbin  sys  usr  
boot  etc  lib  lost+found  mnt  proc  run  srv  tmp  var  
[controller@localhost ~]$
```

Figure 5.3: *Fedora root directory listing*

Exercise: Notice that both of the screenshots above are different even though these are both “stock installs” of two commonly used Linux distros. Take a look at your own Linux operating system you are using and try running the same command. Use the GNOME Activities button to search for the phrase **terminal** and launch it by clicking on the terminal icon. Then type the command **ls /** in the terminal. What is the output you see compared to the two images above? Try it on some other distros and compare the output.

How does this compare to other operating systems? Windows operates on a separate idea of distinct letter drives, C, D, and E for example, each mapping to a discrete disk drives and that is the **root** of each drive. Unix was created before it was even technologically possible to have multiple disk drives. Mac OSX is a different case since it is BSD based, it operates on the same principle as Linux but tries to hide it from you by just giving you icons and finders. You can get terminal access to the commandline and see Mac's Unix heritage too.

5.3.1 POSIX Standard

Why are the two filesystem hierarchies different? The answer goes back to the dawn of commercial Unix in the early 1980s. By 1985 there was a growing concern, especially from Richard Stallman, that the GNU project would suffer a Coup d'état by Unix vendors who would subtly change the way Unix interfaced with software and make GNU software incompatible and therefore impractical for use. Richard Stallman spurred on a nascent IEEE standards group to create **POSIX** (pronounced *pah-zicks*), the **Portable Operating System Interface**.

“This is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems. POSIX defines the application programming interface (API), along with command line shells and utility interfaces, for software compatibility with variants of Unix and other operating systems.¹”

In this way a single or a small consortium of Unix vendors could not “run away” with the market. POSIX ensures a level of interoperability between software across Unix distros that have declared POSIX compliance. The first official POSIX standard was released in 1988, a few years ahead of the creation of Linux (1991). The current version is [POSIX.1-2008](#). It is also referred to by its Open Group Base Specification Issue number, which is 7 or POSIX 7. For more details on the specifics of POSIX and what it does see APPENDIX A at the back of the book.

It is great to have a standard but what exactly does POSIX do? Even in that question the answer varies widely, since Unix was already in use at the time for over 15+ years before a standard was in place. This required the standard to back-define some issues and cave on other issues. POSIX defines at a minimum what a certified Unix based system must support feature and API wise, no more, no less. It does not restrict extra non-POSIX features from being included. In reality there is no POSIX Unix version like there is GPL compliant GNU/Linux distros. Most Linux distros are very POSIX compliant by nature, but very few systems are certified POSIX compatible. For example, POSIX doesn't define a default filesystem other than requiring `/`, `/tmp`, and `/dev` be present.

5.3.2 The Linux Standard Base

Seeing the good that POSIX standardization brought to the Unix world, a similar industry group formed the [Linux Standard Base](#), or LSB. The LSB was put together for two main purposes. First it was hoped that application vendors (Oracle, Postgress, Sun, JBoss, IBM, etc, etc) would certify their products to work across different Linux distros that are LSB certified. Vendors pay a significant amount of money to Microsoft to get drivers certified across versions of Windows for instance. Second was for creating a central Linux identity and reference guide for distros to comply with. The LSB started by extending POSIX and included new items unique to Linux. “*The LSB specifies for example: standard libraries, system commands and utilities that extend the POSIX standard, the layout of the file system hierarchy, run levels, the printing system (including spoolers such as CUPS) and tools like Foomatic, and several extensions to the X Window System.*”² This initial standard body came together and was published in January of 2001.

Most importantly the LSB sought to create an **ABI**, [Application Binary Interface](#). This is different than an API, which assumes that there will be the same standard libraries to code against on every Linux distro. The ABI gives assurance that code **compiled** using a C compiler with the ABI compatibility feature turned on will generate a binary file that will run on all LSB compliant Linux systems. This is similar to a Windows .exe package you create. It runs on all Windows platforms from XP all the way to Windows 10, those operating systems are ABI compliant. The LSB wanted to guarantee this so that major vendors of software would not have to maintain multiple software versions, just maintain one version with ABI compatibility, thus preventing a barrier Linux enterprise adoption.

The dream of a unified Linux standard never really occurred. **No one implements LSB the way it was intended.** This post from the Debian LSB mailing list marked July 3rd 2015 says it all:

¹http://www.opengroup.org/austin/papers/posix_faq.html

²https://en.wikipedia.org/wiki/Linux_Standard_Base

“The crux of the issue is, I think, whether this whole game is worth the work: I am yet to hear about software distribution happening through LSB packages. There are only 8 applications by 6 companies on the LSB certified applications list, of which only one is against LSB ≥ 4 . Amongst the distributions, RHEL 7.0 is LSB4.1, and Oracle 6, RHEL 6.0 and Ubuntu 9.04 are LSB 4³.”

5.3.3 Lennart Poettering’s thoughts on POSIX

Lennart Poettering, the Red Hat engineer and leader of the systemd init system has some things to say about POSIX and his being a Red Hat employee seems to mirror Red Hat’s position on the history of the Linux Filesystem.

“POSIX is really an encapsulation of some choices that various Unix systems made along the way, rather than a body of text that got standardized and then implemented. According to Poettering, Linux should use its position as “market leader” (in the market of free Unix-like operating systems) and try out some new things. If developers don’t force themselves into the constraints of the POSIX API, they could develop some really innovative software, like systemd shows. When these new developments happen to turn out really interesting, other operating systems could eventually adopt them as well.”⁴

“Not having to care about portability has two big advantages: we can make maximum use of what the modern Linux kernel offers these days without headaches – Linux is one of the most powerful kernels in existence, but many of its features have not been used by the previous solutions. And secondly, it greatly simplifies our code and makes it shorter: since we never need to abstract OS interfaces the amount of glue code is minimal, and hence what we gain is a smaller chance to create bugs, a smaller chance of confusing the reader of the code (hence better maintainability) and a smaller footprint.”⁵

“In fact, the way I see things the Linux API has been taking the role of the POSIX API and Linux is the focal point of all Free Software development. Due to that I can only recommend developers to try to hack with only Linux in mind and experience the freedom and the opportunities this offers you. So, get yourself a copy of The Linux Programming Interface, ignore everything it says about POSIX compatibility and hack away your amazing Linux software. It’s quite relieving!”⁶

5.3.3.1 Linux Filesystem Standard Hierarchy

The one useful thing that came out of the LSB is the **Filesystem Hierarchy Standard**, or *FHS*⁷⁸. This is a voluntary standard maintained by the [Linux Foundation](#) that includes a standard hierarchy of directories and optional directories that exists under the **root** in a standard Linux distro. You should *memorize* each directory name and its general function⁹.

Directory	Function
bin	Essential command binaries
boot	Static files of the boot loader that copy the kernel into memory on boot
dev	Device file handles
etc	System configuration files, pronounced “ <i>et-see</i> ” or “ <i>ee-tee-see</i> ”
lib	Essential shared libraries and kernel modules
media	Mount point for removable media
mnt	Mount point for mounting a filesystem temporarily, pronounced “ <i>mount</i> ”
opt	Add-on application software packages
run	Data relevant to running processes
sbin	System binaries added by each operating system for management
srv	Data for services provided by this system
tmp	Temporary files - some distros clear this directory upon reboot (pronounced “ <i>temp</i> ”)
usr	Secondary hierarchy - contains X, KDE or GNOME, plus documentation
var	Variable data - used for storing databases, webserver files, and application logs

³<https://lists.debian.org/debian-lsb/2015/07/msg00000.html>

⁴<http://lwn.net/Articles/430598/>

⁵<https://archive.fosdem.org/2011/interview/lennart-poettering>

⁶<https://archive.fosdem.org/2011/interview/lennart-poettering>

⁷http://refspecs.linuxfoundation.org/FHS_3.0/fhs/ch03s02.html

⁸<http://www.linuxfoundation.org/collaborate/workgroups/lsb/fhs-30>

⁹http://refspecs.linuxfoundation.org/FHS_3.0/fhs/ch03s03.html

Directory	Function
<i>proc</i>	Virtual filesystem created at runtime providing process and kernel information as files.

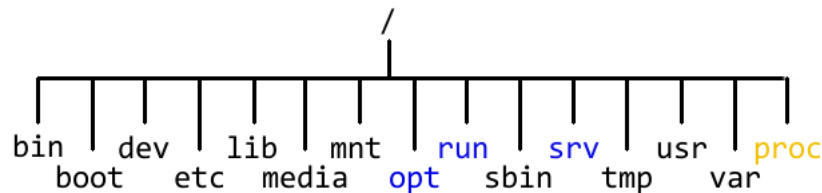


Figure 5.4: *Linux Filesystem Hierarchy Standard - items not bold from above will not be included*

There are 3 additional directories that are non-standard but appear on most Linux distros ¹⁰. You can see a screenshot below.

Directory	Function
home	Users' home directories, containing saved files, personal settings, place that you own.
root	Home directory for the root user - not to be confused with /
lib64	Alternate home for 64 bit libraries

```

controller@controller-VirtualBox: ~
controller@controller-VirtualBox:~$ ls /
bin      home      lost+found  root      tmp
boot    initrd.img  media      run      usr
cdrom   initrd.img.old  mnt      sbin    var
dev     lib       opt       srv     vmlinuz
etc     lib64    proc     sys     vmlinuz.old
controller@controller-VirtualBox:~$
  
```

Figure 5.5: *Ubuntu root full directory listing*

```

controller@localhost:~
File Edit View Search Terminal Help
[controller@localhost ~]$ ls /
bin  dev  home  lib64  media  opt  root  sbin  sys  usr
boot  etc  lib  lost+found  mnt  proc  run  srv  tmp  var
[controller@localhost ~]$
  
```

Figure 5.6: *Fedora root full directory listing*

5.3.3.2 /usr standard subdirectories

There are a series of standard directory locations that are under /usr that you need to know as well.

/usr

Table 5.3: Secondary hierarchy for read-only user data; contains the majority of (multi-)user utilities and applications.

sub-dir	Function
bin	Non-essential command binaries (not needed in single user mode); for all users.

¹⁰http://refspecs.linuxfoundation.org/FHS_3.0/fhs/ch03s02.html

sub-dir	Function
lib	Libraries for the binaries in /usr/bin/ and /usr/sbin/.
local	Tertiary hierarchy for local data, specific to this host.
sbin	Non-essential system binaries, e.g., daemons for various network-services.
src	Source code, e.g., the kernel source code with its header files.
X11R6	X Window System, Version 11, Release 6 (up to FHS-2.3, optional).

5.3.3.3 Red Hat vs POSIX and LSB

Over the years and by tradition inherited from Unix, this is the generally accepted Linux standard filesystem. This layout harkens back to Ken Thompson's original Unix design of nearly ~50 years ago. This means that this structure is wide spread and well known. The downside is a sense of tradition and nostalgia has crept in about this filesystem structure. Note that Red Hat based distros have made a move to change this arguing that when Unix was designed it was constrained by hardware restrictions that no longer exist and should not be maintained for tradition's sake.

Red Hat is arguing that this organization is arbitrary anyway and based on a technology model that doesn't exist anymore. They want to update the filesystem hierarchy but need to maintain backwards compatibility. They may have a point. Some of the application splits between /bin, /sbin, /lib, and /lib64 are completely arbitrary. Red Hat maintains this directory hierarchy but uses symlinks (or a shortcut in the Windows parlance) to their actual location now stored in /usr ¹¹. Red Hat also says this makes Linux applications more in-line with Unix and Solaris, therefore making it easier for customers and companies using Unix and Solaris to port over their software or migrate to Red Hat based distros. By using the command `ls -l /` you can see the light blue colored soft links to the actual directories.

5.4 Path

Remember that our filesystem is an upside down tree? That means we can traverse up and down the tree to arrive at file locations. We now have a way of addressing every single file, folder, and directory on the filesystem. We call this the **path**. This path can be referred to in two ways: **absolute path** and **relative path**. There are also certain locations in the path that have well known names, such as **root** and a user's **home directory**.

5.4.1 Absolute Path and Relative Path

What is the difference? Think of your school or home address. If I told you I live at [324 W 35th Street Chicago Illinois 60616, United States of America](#) that would be my **absolute** address and very helpful, right?

But what if you already lived in the US and in the city of Chicago? What if you lived on the same street as the address listed above, there would be no need to repeat all the detailed information. Perhaps I would just say I live at "*325 West*" and you would know where to go. This concept is called **relative path**. Let us see this concept in action. Later in this chapter we will cover shell commands in detail but for now we will introduce a few basic ones.

- `cd` - used to change directory and move up and down the filesystem tree.
- `pwd` - used to check what your present location is on the filesystem tree.
- `ls` - used to list the content of the directory where you are located on the filesystem tree.

When the terminal window opens you are automatically deposited at a specific location on the filesystem tree. You are placed in your **HOME** directory. In this case the absolute path to that location is `/home/jeremy` (my username being jeremy in this example.) You can always come back to this directory by using the shortcut of `~` like this: `cd ~`. Let us open a terminal window and try this ourselves.

Exercise: Figure out what your `pwd` (present working directory) is. You would do this by simply typing `pwd` on the commandline. Try it and what is the output?

Exercise: The next thing we should do is a listing of the contents in our current directory. We do this by issuing an `ls` command. Try it, what do you see?

¹¹<http://www.freedesktop.org/wiki/Software/systemd/TheCaseForTheUsrMerge/>

Exercise: Let's learn how to use the `cd` command, which allows us to change our current directory location. Let's change our location into the Documents folder, which is under our current `pwd`. Do so by typing `cd Documents`. Once we have changed the directory, type `pwd` and then `ls` command. What do you see?

Exercise: Let's learn about the `cd` command to change our directory location. Type `cd /etc/network` if you are on Ubuntu and `cd /etc/sysconfig/network-scripts` if you are on Fedora or CentOS. Type the `ls` commands—what do you see? Also are these path absolute or relative?

The way to determine is that an absolute path will always start with `/` as the first character. Much like the address example this is telling you the directories absolute location in relation to the **root**. On Ubuntu for instance, the network directory can be found by traversing the tree from `/` down to a system directory called `etc` and then to a sub-directory located in `etc` called `network`. In this case, `network` is a sub-directory of `etc` and contains all the files related to configuring your network card settings.

A relative path assumes the `pwd` portion of the absolute path. From the exercise example above we notice a few things about relative paths. **First** our `pwd` when we open a shell is our home directory. In my case `/home/jeremy`. If we see the hierarchy the directory Documents is under the jeremy directory. the `cd` command takes us down one directory by default. How can we “cd” into that directory? By simply typing `cd Documents`. This is a **relative path**. Why? Notice that there is no `/` in front of the directory name. This tells the `cd` command that you are assuming that the system should start looking for the directory name we passed in the directory we currently are. Could we have typed `cd /home/jeremy/Documents`? Yes we could have, but the first two parts of the absolute path were irrelevant to us because we were already in that location.

How would we navigate back up the tree? What if we want to return to our home directory? There are a few options, assuming that our `pwd` is `/home/jeremy/Documents` which of these would be the best choice?

<code>cd /home/jeremy</code>	absolute or relative path?
<code>cd ~</code>	absolute or relative path?
<code>cd ../</code>	absolute or relative path?

The first two options are absolute paths, the third option is a relative path with something new. Here we are introduced to the **double-dot** operator. The **double-dot** and **single-dot** (sometimes just called *dot*) are the way you can reference locations up the tree or in your current directory (same level). The command `cd ../` is a shortcut to take you **up** one level in the filesystem tree. Double-dots can be chained together to traverse up multiple directories all at once.

Exercise: If you `pwd` is `/home/jeremy/Documents` (replace “jeremy” with whatever your username is) and you want to change directory to `/home`. What command would you use? What happens when you type `cd ../../` on the commandline? What is your `pwd` now?

Exercise: Assume your `pwd` is `/home/jeremy/Documents` and you want to change directory to `/home/jeremy/Downloads` what would be the command to do so in one single command?

You could do this:

1. `'cd ../`
2. and then `cd Downloads` which would get the job done.
3. In one single command: `cd ../Downloads`
4. This command will be interpreted as: `../` means go up one directory
5. The next directory name `Downloads` means go down one directory.

What happens if you try to `cd` into a directory that doesn't exist?

Exercise: You can use the `touch` command to create an empty file called `todo-list.txt` in the `/tmp` directory. Try this command assuming your `pwd` is your Home directory. From the commandline type: `touch /tmp/todo-list.txt`. Now type: `ls`, do you see the file `todo-list.txt`? Why or why not? How would you list (`ls`) the contents of the `/tmp` directory if your `pwd` is `/home/jeremy/Documents`?

```

jeremy@controller-VirtualBox:~$ pwd
/home/jeremy
jeremy@controller-VirtualBox:~$ ls
Desktop  Downloads  Music  Public  Videos
Documents examples.desktop Pictures Templates
jeremy@controller-VirtualBox:~$ cd Documents/
jeremy@controller-VirtualBox:~/Documents$ pwd
/home/jeremy/Documents
jeremy@controller-VirtualBox:~/Documents$ cd ../Downloads/
jeremy@controller-VirtualBox:~/Downloads$ pwd
/home/jeremy/Downloads
jeremy@controller-VirtualBox:~/Downloads$

```

Figure 5.7: Path demonstration

```

jeremy@controller-VirtualBox:~/Documents$ cd topsecret-documents
bash: cd: topsecret-documents: No such file or directory
jeremy@controller-VirtualBox:~/Documents$ █

```

Figure 5.8: File or Directory not found

5.5 The Linux Shell

5.5.1 Shell, Terminal, and Commandline

If you remember from chapter 4 that the word “terminal” came from the actual DEC VT-100 physical terminals that were in use in the late 70’s and early 80’s for use in interfacing to Unix systems. A terminal is a way to display text in 80 by 25 line screens of text. The **Shell** is the actual application that runs on the terminal to display that screen of text and allow the user to interact with the Kernel via commands. There are many types of shells that have different features. Shells give the user the ability to create scripts for executing multiple commands, command completion, command history, command aliases, and more. You open a terminal and in a terminal session you use a shell. You can have multiple terminal sessions and use different shells on each one if you prefer. In total the combination of using a terminal to access a shell can be called using the *commandline*. The diagram below shows how the commands you type in the terminal are processed by the shell, given to the kernel and then output is returned to the user.

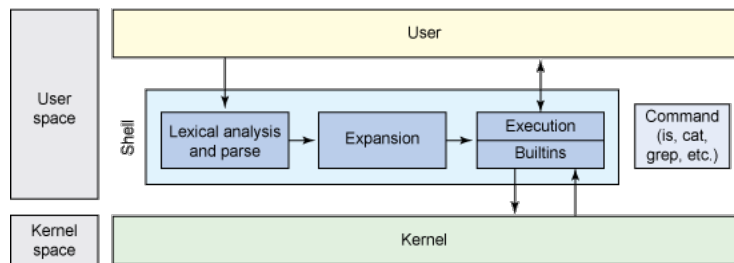


Figure 5.9: Shell/OS interaction diagram

5.5.2 Linux Shell Commands

Many may say, “Hey I have a nice point and click GUI why do I need to use the crusty old commandline?” That is a valid question. In reality the GUI is not separate from the shell. It is a “visual sugar” built on top of the Shell. Anything you click on in a GUI in reality is in reality **executing a command through the shell**. In most cases using the Shell has more features for your command than in the GUI. The GUI by definition cannot have more capability than the shell.

5.5.3 Basic Shell Commands

The Shell is where we enter commands for navigation, manipulation, and creation of text files. Some of the basic commands covered in chapter 5 and 6 are in the table below. In the course of your career you need to memorize at least these commands by name and what they do.

meta-commands	cd	ls	pwd	file	date		
file-related	cp	mv	mkdir	touch	wget	rmdir	rm
file-ownership	chmod	chown	chgrp				
display-related	cat	less	more	tac	head	tail	
display-augmentation	grep	awk	sed				
help commands	man	info	find	which	locate		

cd This command is used to help you change the directory location. The shell can only be pointed at one file location at a time. In a GUI you are used to “*clicking*” on folders to navigate up and down the filesystem hierarchy. What you are actually doing when you click is issuing **cd** commands to the kernel which is returning instructions of what the GUI should now render.

```
cd /tmp
```

ls This command is used to list the content of the current directory you are in.

```
ls /etc
```

cp This command is used to cp the contents of a file, can also be used to copy and rename a file.

```
cp *.txt /home/user/Documents
```

mv This command is used to rename or move a file in-place, though the actions are technically the same. By renaming a file you are essentially moving its contents to a new file in-place.

```
mv ./secret-passwords.txt ./new-secret-passwords.txt
```

mkdir This command is used to create or make a new directory. Pronounced “*make-dur*”

```
mkdir ~/legally-downloaded-music
```

touch This command is used to create a new blank file or to update the timestamp of an existing file without opening it.

```
touch todolist.txt
```

cat This command is used to display the content of a file to the screen. Technically created to concatenate two files but will accept “nothing” as one input and concatenate nothing plus a filename content and output that to the screen. There are lengthy articles discussing if this is the [correct method or a superlative way](#). We will use it through out the textbook because of readability and common usage acceptance.

```
cat /etc/apache2/sites-available/000-default
```

tac This command is used to display the entire content of a file but backwards. The last line of the file will be printed first

```
tac /etc/passwd
```

```
# The contents of tac can be piped to a less command to allow paging:
```

```
# hit the letter 'q' to exit the less command
```

```
tac /etc/passwd | less
```

less This command is a *hack* on the **more** command which allows for scrolling the text of a file using the enter key or page down, and space bar or arrow down key for single line scroll. The **more** command only went down, not up. Hence the more powerful command **less** which pages and line scrolls up and down a document. Less is more.

```
less /var/log/apache2/error.log
```

date This command displays the common date and time format. It has full formatting customization so you can display date and time exactly as you need to.

```
date '+%m%d%y'
```

file This command has been in Unix since at least 1973 records show. Its use is to help you identify what type of file something is. It will out put the type of file if it is text, a directory, a binary, or something else.

```
file self-destruct
```


pwd This command tells you your **present working directory** on the Linux filesystem tree.

`pwd`

wget This command is used to retrieve files over the internet via an HTTP GET request. If you are using the commandline you cannot “click” on a link but when you click on a link you are sending an HTTP GET request in reality. This gives you control without having to leave the commandline. The sample below will you download the markdown files used to build this textbook.

`wget https://github.com/jhajak/Linux-text-book-part-1/archive/master.zip`

unzip This command is used to extract a compressed file of type zip. There is a GNU version of this named `gzip` that extracts `gzip` files.

`unzip music.zip`

tail This command is used to display the last 10 lines of a file. The number 10 can be overridden by using the `-n` flag to display an arbitrary number of lines.

`tail /etc/passwd`

`tail -n 25 /etc/passwd`

man This is the command you use when you can’t remember how to use other commands. just type the word `man` and then any other command name and you will automatically drop into a **less** session which will explain the use and history of that command. This is a very helpful tool as no one can memorize every single command. From here on out in the class when you ask a question, the instructor’s first answer will be, “*Check the man page.*” As you progress in your Linux journey the **man** command will be your best friend—use it wisely young Jedi.

`man wget`

Exercises: Let’s create a text file that has the current date and time in it. Type `date > timestampfile` (The `>` is a shell meta character that redirects output from the screen to another file. We will learn more about this in chapter 6.)

Exercise: Now copy that file to the `/tmp` directory—what would be the commands? Now let us make a new directory under our home directory called `notes` (remember the filesystem is an upside down tree)

Exercise: What command would we use? `mkdir notes`. Now let us copy the file `timestampfile` into the `notes` directory. What command would you use? What would the full command and arguments look like? How would you change directory and list the content of the `notes` directory (hint `cd` and `ls`)? *Note* if you get lost in the Linux filesystem tree you can always type `cd ~` (tilde) and you will be taken back to your user home directory.

5.5.4 Command Nomenclature

Commands contain three parts in this order:

- command or binary name (required)
- options (may or may not be required)
- arguments (may or may not be required)

Let’s take this command for instance:

`ls -la /etc`

The first two letters of the command listed above `ls` make up the command for listing the contents of a directory. The command must be followed by a space. The next letters are preceded by a **dash** to tell the shell interpreter that these letters are **options**. Options are usually single letter representations of functionality. The `-l` options tells the `ls` command to give a long listing of a directory with details and the “`-a`” option tells the shell to print all files in the directory, including hidden files. Options can be combined in most cases into a single string preceded by a dash. So `-la` can also be written as `-l -a`. Additionally there are options that use full English language words instead of single letters preceded by **two dashes**. You can use the `man` command followed by the name of the command in question to see all of it’s usage options.

The final value of `/etc` in the command `ls -la /etc` is an argument passed to the `ls` command telling the `ls` command to list the contents of the `/etc` directory. If this value is left empty the shell assumes you mean the `pwd` or your current location in the filesystem.

```
File Edit View Search Terminal Help
[controller@localhost ~]$ ls /home/controller
Desktop Documents Downloads Music Pictures Public Templates Videos
[controller@localhost ~]$
```

Figure 5.10: Listing of the `/home/controller` directory

```
total 88
drwx-----. 16 controller controller 4096 Sep 13 00:29 .
drwxr-xr-x.  3 root      root      4096 Aug 30 23:26 ..
-rw-----.  1 controller controller   5 Sep 12 22:37 .bash_history
-rw-r--r--.  1 controller controller  18 Jan 23 2015 .bash_logout
-rw-r--r--.  1 controller controller 193 Jan 23 2015 .bash_profile
-rw-r--r--.  1 controller controller 231 Jan 23 2015 .bashrc
drwx-----. 17 controller controller 4096 Sep  9 10:54 .cache
drwx-----. 13 controller controller 4096 Aug 31 02:26 .config
drwxr-xr-x.  2 controller controller 4096 Aug 31 00:01 Desktop
drwxr-xr-x.  2 controller controller 4096 Aug 31 00:01 Documents
drwxr-xr-x.  2 controller controller 4096 Aug 31 00:01 Downloads
-rw-----.  1 controller controller  16 Aug 31 00:01 .esd_auth
drwx-----.  2 controller controller 4096 Sep 13 02:22 .gnupg
-rw-----.  1 controller controller 1550 Sep 13 00:29 .ICEauthority
drwx-----.  3 controller controller 4096 Aug 31 00:01 .local
drwxr-xr-x.  5 controller controller 4096 Aug 31 02:04 .mozilla
drwxr-xr-x.  2 controller controller 4096 Aug 31 00:01 Music
drwxr-xr-x.  2 controller controller 4096 Aug 31 00:01 Pictures
drwxr-xr-x.  2 controller controller 4096 Aug 31 00:01 Public
drwx-----.  2 controller controller 4096 Aug 31 02:26 .ssh
drwxr-xr-x.  2 controller controller 4096 Aug 31 00:01 Templates
drwxr-xr-x.  2 controller controller 4096 Aug 31 00:01 Videos
[controller@localhost ~]$
```

Figure 5.11: Long and hidden Listing of the `/home/controller` directory

5.5.5 How to Read Shell Commands and “Speak Linux”

In working with many excellent people over the years I have also found that there is a common Linux spoken language. When talking to others, you find that for the most part the standard Linux filesystem has been memorized. As well as the most common Linux tools through repeated usage. With this in mind you can “speak” a command without mentioning certain aspects—they are implied.

Let’s take a simple command to list the contents of the `/etc` directory with a long listing and showing all, including hidden files for instance. How would you verbalize the command to your co-worker?

```
ls -la /etc
```

You would say “*el-es el-aaa eee tee cee*”. Note I didn’t mention any dashes or slashes. Why? Because the context of the `ls` command tells me that the next characters listed are options belonging to the `ls` command. The shell assumes that you are giving an absolute path because of the slash preceding `/etc`.

Exercise: How would you read and say this command?

```
rm -rf /tmp/topsecret
```

Exercise: How would you read and say this command?

```
mkdir -p /mnt/datadrive
```

Exercise: How would you read and say this command?

```
ls -la /etc/network/interfaces
```

5.5.6 History of the Shell

On modern Linux there is one standard shell in place on pretty much all major Linux distros, the [GNU Bash Shell](#). It was created in parallel with the rise of Linux—the two are practically tied together. But to understand how we got to the modern Bash Shell we have to go back to the beginning. As always the history of anything on Unix/Linux goes back to Ken Thompson ¹². By 1972 there was a single shell in Unix in use, referred to by others as the [Thompson Shell](#). This shell was written by Ken Thompson and was written for Ken Thompson solving the technical problems he had back in 1972. By 1979 computer processing power and the nature of computing tasks were light years away from where they had been when Thompson started his shell. “*The shell’s design was intentionally minimalistic; even the if and goto statements, essential for control of program flow, were implemented as separate commands*” ¹³. A new shell was needed.

Two almost simultaneous efforts on different sides of the country were happening to replace the Thompson shell. Remember that Unix had split into BSD Unix out at Berkeley California and AT&T Unix which was the commercial Unix back east. The BSD group was developing the C shell (csh) under Bill Joy. AT&T researcher Steven Bourne was developing the Bourne Shell (sh).

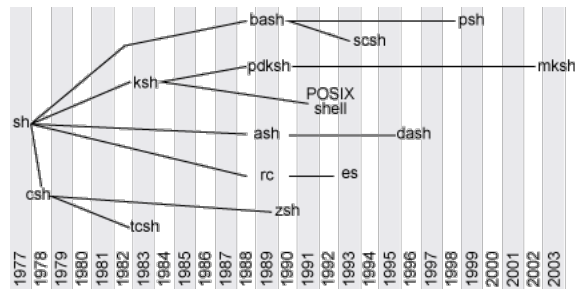


Figure 5.12: *Timeline history of Unix/Linux Shells*

5.5.6.1 C Shell

The **C Shell** was written primarily by [Bill Joy](#) while working on the BSD Unix distro in 1978/79. Joy concluded that since Unix was written in the C language and most programs at the time were written using C, it didn’t make sense to change the language one was using in their shell to something other than C. So Joy implemented the shell concept Thompson had started with vastly improved features and using the C language based syntax. By 1984 there was an improved C Shell called the tcsh that fixed the deficiencies of the original C Shell and added features. This was only available on some BSD variants at the time.

Criticism of C Shell: “*Although Stephen Bourne himself acknowledged that csh was superior to his shell for interactive use, it has never been as popular for scripting. Initially, and through the 1980s, csh could not be guaranteed to be present on all Unix systems, but sh could, which made it a better choice for any scripts that might have to run on other machines. By the mid-1990s, csh was widely available, but the use of csh for scripting faced new criticism by the POSIX committee, which specified that there should only be one preferred shell, the Korn Shell, for both interactive and scripting purposes. The C shell also faced criticism from others over the C shell’s alleged defects in syntax, missing features, and poor implementation.*” ¹⁴

5.5.6.2 Bourne Shell

Steven Bourne was a researcher at AT&T Bell labs implemented his own shell to replace the Thompson Shell in 1977. The Bourne Shell was distributed in standard Unix from Version 7, SystemIII, SVR2, SVR3, SVR4 ¹⁵.

- Built-in test command – System III shell (1981)
- # as comment character – System III shell (1981)
- Colon in parameter substitutions “`${parameter:=word}`” – System III shell (1981)
- Functions and the return builtin – SVR2 shell (1984)

¹²<https://www.ibm.com/developerworks/library/l-linux-shells/>

¹³https://en.wikipedia.org/wiki/Thompson_shell#cite_note-Mashey1976-10-13-1

¹⁴https://en.wikipedia.org/wiki/C_shell#Criticism

¹⁵https://en.wikipedia.org/wiki/Bourne_shell#Features_introduced_after_1979

- Built-ins unset, echo, type – SVR2 shell (1984)
- Source code de-ALGOL68-ized – SVR2 shell (1984)
- Modern “\$@” – SVR3 shell (1986)
- Built-in getopts – SVR3 shell (1986)
- Cleaned up parameter handling allows recursively callable functions – SVR3 shell (1986)
- 8-bit clean – SVR3 shell (1986)
- Job control – SVR4 shell (1989)
- Multi-byte support – SVR4 shell (1989)

5.5.6.3 Korn Shell

By the start of the 1980s you had two shells, Bourne Shell and the C Shell. Both had strengths and weaknesses as detailed above. There is an American expression, “The grass is always greener on the other side of the fence.” The C shell users wanted features from the Bourne Shell and vis-a-versa. Another researcher at Bell Labs, named David Korn, began the process of starting a new shell that was backwards compatible with the Bourne Shell but added the newer features of the C Shell. This was called the [Korn Shell](#) (ksh) and released in 1983. The Bourne Shell was standard on all AT&T Unix so any new shell had to guarantee backwards compatibility otherwise that shell was never going to receive adoption. The improvements the Korn Shell made were apparent and it was soon included on AT&T Commercial Unix as the standard shell. In 1991/1992 the POSIX standard adopted KSH as the standard shell that any Unix system claiming POSIX compliance was required to have. By 2005 the Korn Shell was opensourced under a free license but it was too late.

5.5.6.4 GNU Bash Shell

The ksh was a suitable replacement for the Bourne Shell and C Shell, but from the Free Software Foundation’s view there was one main problem. All of these shells were under proprietary AT&T licenses. This led Richard Stallman and the FSF to fund development seeing that having a free software licensed shell was central to their dream of a completely free operating system. Work was begun in 1989 and completed in 1991. The shell was named the Bash Shell. It was a clever hack on the Bourne Shell name. Bash was a superset of the Bourne Shell and feeling as though they were “freeing” the Bourne Shell from its past life of closed software, they named it the “*Bourne Again Shell*”–Bash. Bash is the GNU Project’s shell. It is intended to conform to the IEEE POSIX P1003.2/ISO 9945.2 Shell and Tools standard. It offers functional improvements over sh for both programming and interactive use. In addition, most sh scripts can be run by Bash without modification ¹⁶.

The improvements offered by bash over sh, csh, and ksh include:

- Command line editing
- Unlimited size command history
- Job Control
- Shell Functions and Aliases
- Indexed arrays of unlimited size
- Integer arithmetic in any base from two to sixty-four

5.5.6.5 Open Source sh Replacements

Unlike Bash which provided a “free” sh compatible shell for the GNU system, there was much code written in sh but no opensource version available. In 1989 Kenneth Almquist released the Almquist Shell (ash) which was an updated and opensourc version of the Bourne shell. In 1995/96 Debian maintainers released another improved version of Ash Shell called the Debian Ash Shell (dash) and this is the standard replacement for the traditional `/bin/sh`.

5.6 File Permissions and Ownership

Seeing that everything in Linux is file, there is a simple security model. There are three types of permission per file: **read**, **write**, **execute**. These files give a combination of permissions. With **read permission** you can display the content of a file or copy it, but cannot delete or rename it. For that you need **write permission**. If a file is a shell script or an executable binary you will need **execute permission** for it to run. How can you tell what permissions a file has? Type `ls -l` see the output. We have seen this output before and now we will explain it.

¹⁶<http://www.gnu.org/software/bash/>

difference between absolute and relative path and a number of the basic shell commands. Finally we covered file permissions.

5.7.1 Review Questions

- 1) What is the numeric value of a file with the permissions `rwxr-r-`?
 - a. 777
 - b. 700
 - c. 766
 - d. 744
- 2) What is the numeric value of a file with the permissions `rw---`?
 - a. 700
 - b. 400
 - c. 600
 - d. 007
- 3) What is the numeric value of a file with the permission `rwxr-xr-x`?
 - a. 711
 - b. 755
 - c. 644
 - d. 227
- 4) What is the name of the command you use to list the contents of a directory?
 - a. `lst`
 - b. `less`
 - c. `cat`
 - d. `ls`
- 5) What is the name of the command you use to display the content of a file to the screen?
 - a. `dg`
 - b. `tac`
 - c. `ls`
 - d. `cat`
- 6) What is the name of the command you use to display the content of a file to the screen that allows you to page up and down?
 - a. `more`
 - b. `less`
 - c. `ls`
 - d. `page`
- 7) Every shell command has three components:
 - a. Command, arguments, flags
 - b. Command, options, arguments
 - c. Commands, arguments, options
 - d. Commands, flags, options
- 8) What does the user use to issue commands to the kernel?
 - a. Terminal
 - b. Commandline
 - c. Magic
 - d. Shell
- 9) What is the name of the GNU shell that is standard across all Linux Distros
 - a. `ksh`
 - b. `sh`

- c. `ssh`
 - d. `bash`
- 10) Based on the `ls` command, what is the option to do a long listing?
- a. `ls --all`
 - b. `ls -n`
 - c. `list`
 - d. `ls -l`
- 11) What is the command you can use to find out additional usage information about a shell command?
- a. `about`
 - b. `man`
 - c. `F1`
 - d. `/?`
- 12) Which of these directories is not part of the LSB LFH?
- a. `bin`
 - b. `media`
 - c. `temp`
 - d. `opt`
- 13) The Linux Filesystem is an upside down what?
- a. `root`
 - b. `object`
 - c. `tree`
 - d. `mess`
- 14) What is the character name of the top of the Linux Filesystem tree?
- a. `/`
 - b. `./`
 - c. `../`
 - d. `slashdot`
- 15) Everything (directories, files, devices) in Unix/Linux is a what?
- 16) What is the name of the Unix system standard developed in ~1985 that defines at a minimum what a certified Unix based system must support?
- a. `Xenix`
 - b. `LSB`
 - c. `POSIX`
 - d. `Linux`
- 17) What is the directory where configuration files are stored in Linux?
- a. `bin`
 - b. `sbin`
 - c. `etc`
 - d. `conf`
- 18) What is the directory where all the essential command binaries are stored?
- a. `bin`
 - b. `sbin`
 - c. `usr/sbin`
 - d. `usr/bin`
- 19) What is the absolute path of a user's home directory, assuming the user is named controller?
- 20) True or False - Lennart Poettering created POSIX

5.7.2 Podcast Questions

Command Line Heroes: Bash

- ~0:20 Who is the creator of the Bash Shell?
- ~0:43 Which organization did the creator of the Bash Shell write the shell for?

- ~2:05 How does the podcast host define a shell script?
- ~2:23 Shell scripts are the key to what?
- ~3:28 When did Ken Thompson release his shell and what was it missing?
- ~3:45 What year was shell scripting come into existence?
- ~4:27 What was the shell that became the AT&T UNIX standard shell?
- ~5:53 The Bourne Shell was licensed and owned by whom?
- ~7:59 Why was Brian Fox the perfect person to develop the Bash Shell?
- ~9:30 How long did it take to create the Bash shell and what was difficult about this?
- ~12:02 What did Brian accidentally do to the Bash Shell?
- ~14:48 What was the other shell released one month before Bash?
- ~15:19 When was GNU Bash released?
- ~18:40 What was the released/intended purpose of GNU Bash?
- ~19:25 What words and terms are now in use in everyday English?
- ~20:46 Was Steven Bourne “cool” with the Bash Shell?
- ~22:52 What prepares you to be more of a long-term thinker?

5.7.3 Lab

The objectives of this lab are to use the shell commands we learned in this chapter and understand their proper usage and form. The outcome will help your ability to successfully use the Linux Shell for navigation, file creation, and file modification. Resist the temptation to use the GUI file manager and a web browser. All actions will be done through the shell unless noted.

Note: All work should be done on either Ubuntu desktop or Fedora desktop unless noted and using the Terminal.

The following commands are intentionally out of order as you will be issuing commands to fix the order.

- 1) Opening a terminal, `cd` to the **Documents** directory. Issue the `pwd` command to find your present working directory (or current location)
- 2) Still in the **Documents** directory, use the `mkdir` command to create a directory named: **itmo356**
- 3) `cd` into the itmo356 directory. Issue the command to show your present working directory. Create the below hierarchy of sub-directories. Once created take a screen shot of the output of the `ls` command in the distros directory
- 4) distros > BSD
- 5) distros > Debian (yes this spelling should be capital)
- 6) distros > redhat
- 7) Inside of each of the three directories just created we will need to create some text files. Take a screenshot of the `ls` listing the contents of each directory showing how to create the files and listing them in the directory.
- 8) In the directory BSD use the command to create a file: Create three files names: oracle-linux.txt opensbd.txt netbsd.txt
- 9) In the directory Debian use the command to create a file: Create three three names: ubuntu.txt freebsd.txt devuan.txt
- 10) In the directory redhat use the command to create a file: Create three files names: fedora.txt rhel.txt debian.txt
- 11) At the same level as distro distros level -> omnios.txt
- 12) In the text files created you will notice that a few of them are out of place, such as a debian.txt is in the redhat folder. Issue the command to move the text files to the correct locations and show a directory listing before and after the move command. (*Hint: ls mv ls*)
- 13) Use the `mv` command to rename the Debian directory to have a lowercase “D”
- 14) Use the `rm` command to delete oracle-linux.txt file
- 15) Use the `mkdir` command to create a directory at the distros level named: **illumos** and move the omnios.txt file into **illumos**
- 16) `cd` to the distros BSD directory - what would be the command using a relative path to change your `pwd` to redhat?

- 17) From the redhat directory - what would be the command using the relative path to change your pwd to your home directory?
- 18) From your home directory - what would be the command using the relative path to change your pwd to the debian directory created earlier?
- 19) From the debian directory - what would be the command using an absolute path to change your pwd to /etc/ssh/ssh_config.d?
- 20) Use the `man` command to figure out how to display the format of the `date` command as month, day, year combined to look like: 10152021
- 21) Using the file on Ubuntu 20.04: `/var/log/syslog`: Type the command to display the last 10 lines of this file
- 22) Using the file on Ubuntu 20.04: `/var/log/syslog`: Type the command to display the first 10 lines of this file
- 23) Using the file on Ubuntu 20.04: `/var/log/syslog`: Using the `man` command on the previous question, find how to display the last 25 lines of this file
- 24) Using the file on Ubuntu 20.04: `/var/log/syslog`: Type the command to display the entire contents of this file backwards (last line to first)
- 25) Using the file on Ubuntu 20.04: `/var/log/syslog`: Use one of the commands to allow for paging through one of these commands (hit 'q' to quit the paging command)
- 26) Clone the textbook source code into your virtual machine. `cd` into Documents and issue the command: `git clone https://github.com/jhajak/Linux-text-book-part-1.git`. (*Hint*: You may have to use your distro package manager to install the `git` tool)
- 27) Issue the `cd` command to change directory into `Linux-text-book-part-1`. Issue the command to display what type of file `./title/metadata.yaml` is.
- 28) Issue that same command to display what type of file `Appendix-A` is.
- 29) Using in the book source code, under files > chapter-05 > sample-script: copy the file `sample-command` to your home directory. Use the command to give the script execute permission `+x`. Issue the `ls -l sample-script/sample-command` command to show the permissions of just the `sample-command` file
- 30) Move the file `sample-command` to the location `/usr/local/bin` (**Note**: you will need to add the `sudo` command in front to give yourself root privileges to move a file to this location)
- 31) Then from the command line (any location) execute the command: `sample-command`, and if successful you will receive a message of success
- 32) Execute the `ssh-keygen` command on the command line (from any location) and accept all the default values (just hit enter for now, we will cover RSA in depth later in the text). This command generates two files that are part of an RSA keypair, located in `~/.ssh`
- 33) `cd` into that directory and type the command to show the long listing of the directory
- 34) From the textbook sample code > Files > Chapter-05 > sample-scripts > copy the file `date-time-script.sh` to your home directory. `cd` to your home directory and execute the shell script you just copied (which will print out the current datetime) with the command: `./date-time-script.sh`. You get an error message stating **permission is denied**: Explain why?
- 35) Type the command to enable your user to be able to execute this command and capture the results
- 36) From the textbook sample code > Files > Chapter-05 > sample-textfiles > `distro-list.txt` : issue the command to display the content of the `distro-list.txt` to the screen
- 37) Issue the command using a relative path to copy the file `distro-list.txt` to `distros` directory you made under the `itmo356` folder
- 38) Issue the command to show the listing of the `distros` directory content and showing that the file was copied
- 39) Use the `wget` command to retrieve a copy of the Packer.io binary for Linux. Use this URL as the argument for `wget` https://releases.hashicorp.com/packer/1.7.6/packer_1.7.6_linux_amd64.zip
- 40) Use the `unzip` command to unzip the binary and extract the packer executable. Issue the move command, `mv` to move the executable to this location: `/usr/local/bin`. **Hint**: you will need to use `sudo` to get the permission needed for this operation and you may need to install the `unzip` application using your package manager. To show this was successful take a screenshot of the output of the command `packer -v`.

5.7.3.1 Footnotes

Chapter 6

Shell Meta-Characters, Pipes, Search and Tools

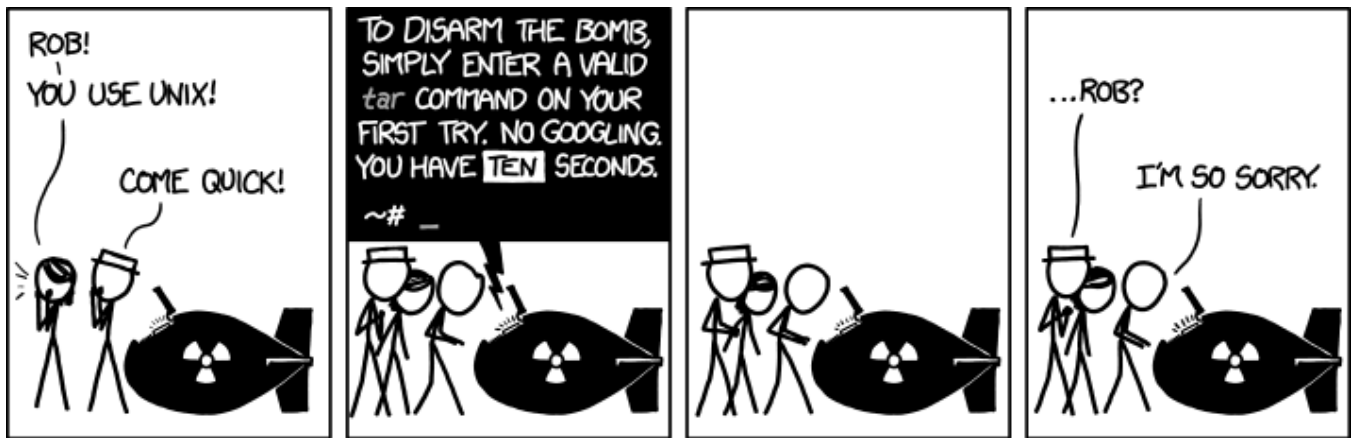


Figure 6.1: *The beauty of Unix Commands*

6.1 Objectives

In this chapter we will be continuing our exploration of the commandline. We will be expanding our experience of the original paradigm that Thompson and Ritchie envisioned when designing Unix.

- Understand the nature of shell meta-characters and how they enhance the shell capabilities
- Understand the concept of standard in, standard out, and standard error
- Understand the concept of input/output redirection and piping
- Understand how to search the file system for files and directories
- Understand how to use the `grep` command for detailed search and replace options using shell meta-characters

6.2 Outcomes

At the conclusion of this chapter you will have a definite understand of the Linux shell and its utilities. You will know the nature of shell meta-characters and how they can enhance the capabilities of shell commands. You will be able to use the concepts of standard input, standard output, and standard error to redirect output as you need it. You will understand the meta-characters used for input redirection and the concept of “|” called piping - that enables single commands to send their standard output as standard input to another command. We will explore the `find` and `locate` commands that are used to find and filter files on the system. We will use the `grep` tool for find

and replace options and advanced parsing of file content beyond what `find` and `locate` can do. Finally we will use compression tools and for creating archives and for extracting them.

6.3 Shell Meta-Characters

In the last chapter we learned about the Linux shell and its purpose to help the user interact with the kernel. We learned that the GUI is just a *syntactic sugar* layer sitting on top of the shell. We also learned a series of essential commands in order to create and manipulate the contents of our file system. The next layer within the shell to be introduced is something called **shell meta-characters**. When Ken Thompson was creating Unix and the first user shell, he quickly realized a need to be able to perform certain repetitive tasks. For example, the concept of using wildcards to do pattern matching.

Or to use negation or ranges to look for particular filenames or directories. One of the advantages of making everything in Unix a file is that a file only has one data type, and that is of type **text**. Since everything is text, any file or directory can be searched for or filtered based on text patterns. Enter shell meta-characters which extend the normal capabilities of the shell. If you wanted to list the contents of any directory starting with the letters “Do”, how would you do it? By adding a wildcard to an `ls` command like this: `ls -l Do*`. This command reads a directory’s content and feeds it any filenames that match the first two characters “Do” and any number of other characters, (including 0 characters), represented by the star or asterisk (*). Shell meta-characters replace having to write a C program to recreate this same functionality. There are 18 characters that are standard across the bash shell¹—common punctuation/non alphanumeric characters that were adopted to represent these common repetitive text modification tasks done in the shell.

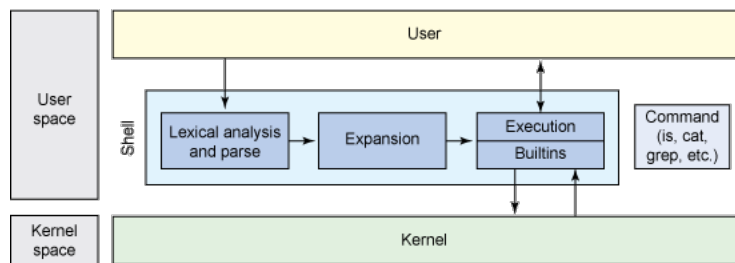


Figure 6.2: *User -> Shell -> Kernel -> Shell -> User*

In the above diagram, the box labeled *Expansion*, is where the shell *expands* any meta-characters into text that can be interpreted by the shell. The following is a list of the major shell meta-characters you need to know. More information and examples can be found at the [Linux Documentation Project’s](#) website.

&& The double ampersand character (shift+7) command allows you to execute two or more commands together in sequence. Note the `&&` requires each successive command to return successfully or else the entire string will stop at the command that failed. **Usage example:**

```
unzip -d book master.zip && cd book && cat Readme.md
```

; Unlike the ampersand character the semi-colon “;” allows you to chain commands together that will execute in sequence regardless of the previous commands return status. The example command will execute and return two errors telling you the file or directory doesn’t exist, but the third command will execute showing the data. **Usage example:**

```
ls /topsecret; cd /topsecret; date
```

| This character is called the “pipe” because it looks like a vertical bar or a piece of a pipe. It serves the function of connecting the output of one command to the input of another command—not unlike a pipe under your sink. The character is typed by pressing the shift+the key located directly above the enter key. In this example here we display the content of the `chapter-05.md` and *pipe* it to the `grep` command which filters the file showing us only the lines containing the term *permission*. **Usage example:**

```
cat Chapter-05/chapter-05.md | grep permission
```

¹http://pubs.opengroup.org/onlinepubs/009695399/basedefs/xbd_chap09.html

- * The asterisk (shift+8) is the wildcard representative. It can be used in any shell command when you want to let the computer to the work or when you are not quite sure of a files spelling. **Usage example:**

```
ls *.md
cat Linux-*.pdf
rm -rf ./*
```

- ? The character question mark is a single character wildcard. Multiple question marks can be combined but are single wildcards. ??? .txt will list all files with three characters and the .txt extension. **Usage example:**

```
ls memo?
ls vegetable-?-report.txt
ls ??? .txt
```

- ' single tic or single quote. Any characters or variables surrounded by a single tic will be interpreted literally. Unlike the previous example this command will only print the literal characters \$DT not expand the variable's contents. The first item below will not work, why? You will see a » on the next line which is a continuation. The shell interpreter is confused and matches the first two " but the second is not closed. See the next example below. **Usage example:**

```
echo 'Today's date is $DT'
echo 'Today the date is: $DT'
```

- " double tic or double quote. Any characters or variables surrounded by double tics will interpret shell variables that inside of the string before passing the contents to a command. This command will print the content of the date command from the previous example along with some text to the screen. You can include single and double tics in a single command but make sure they are matched. **Usage example:**

```
echo "Today's date is $DT"
```

- [] Square brackets indicate sets or ranges of characters to be processed. In a sense it is a mini-for loop for processing files names. We can present multiple character sets or if we use a - we can present a numeric or alphabetic range of values to be passed to a shell command. **Usage example:**

```
ls file[24]
rm file[1-4]
ls file[!a-z]
```

- () Parenthesis are used to start a sub-shell. The parent shell doesn't have memory access to this sub-shell. We will cover this more in detail in the next chapter. **Usage example:**

```
(a=hello; echo $a)
```

- < > » Angle brackets or less than greater than - allow for input and output to be redirected. Think of them as arrows pointing. Single arrow is destructive meaning if the filename exists the previous contents will be destroyed and the file recreated. Double arrow is an append operation and will append the content to the end of the filename given as an argument. The arrows can be combined with the & to help suppress output and that will be described in the next section in detail. Note how the normal output to the screen is being redirected to these files by the arrow operators. **Usage example:**

```
date > /tmp/todaysdate.txt
date >> /tmp/todaysdate.txt
echo "Buy milks and eggs!" >> ~/Documents/my-shopping-list.txt
```

- { } Curly braces are expansion braces for declaring variables with variables in them. Note the difference. The first command executes the content of whoami command and returns it and passes it to the echo command. The second example allows you to use brace expansion to create multiple directories at once chapter-01-09. **Usage example:**

```
echo ${username-`whoami`}
```

```
mkdir chapter-0{1..9}
```

Pound sign or hash is used to comment out the lines in a shell script. We will cover this in detail in the next chapter.

^ The caret (shift+6) is a meta-character used in conjunction with the [] range brackets in order to tell the shell to negate the range included in the square brackets. `ls [^r]*` will find file1-file4, but will not list the files named rfile1-rfile4. **Usage example:**

```
ls [^r]*
ls [^b-d]
```

! Exclamation or pronounced as *bang*. This is used to negate a test condition. This will be covered more in the next chapter when we cover shell scripts.

~ The tilde (shift+the key to the left of the number 1) is a shortcut representing your user's home directory. `~-` is a combination that references the previous directory you were in. **Usage example:**

```
cd ~
cd ~/Documents
cd ~-
```

<tab> Tab key is used for autocomplete on the commandline. This is a feature of bash, not present in `sh`, `cs`, and `ksh`.

\ By placing a \ in front of any character, the shell will *escape* that characters function. This is helpful if you want to print out the text of commands without having the shell execute them.

```
echo "Assign date command output to a variable type: DT=`date`; echo \${DT}"
echo "This is a variation of the above command: DT2=${date}; echo \${DT2}"
```

\$ The dollar sign (shift+4) is used to reference shell variables. You can declare a local variable by assigning a value with the equal sign—just like you can in C or Java. But in this case space matters (no spaces) and you only use the \$ sign the reference that variable. There are a number of preset system environment variables that the os sets for you and you can make your own as well. The command `printenv` will display all of the system environment variables. By convention shell variables are usually declared all capital letters. Note that shell variables only “live” for the time that the terminal window in which they were declared is open.

```
echo $PATH
DT=`date`; echo $DT
DT=${date}; echo $DT
JAVA_HOME="/usr/bin/java"; echo $JAVA_HOME
```

There is a final meta-character called the backtick or formally the grave accent. The backtick key is to left of number 1. The backtick is used for encasing Linux binary command names. The backtick tells the system to interpret the command and execute it and return its value to the user for further processing. In the 2 prior examples we stored the content of the date command to a shell variable named DT. **Usage example:**

```
DT=`date`
# second method to execute the above
DT2=${date}
CONTENTS=`cat /etc/services`
```

6.4 Standard input, output, and error

The original concept behind defining standard input and output devices came from the idea that Unix grew over the course of 40 years of output and input technology. Originally there were no keyboards—just teletypes. Originally there were no screens just ttys to print out on paper. Then came *glass ttys* and terminals. Then came Line and Dot Matrix printers. Standard keyboards via the ISA input bus were invented. At one time on a busy shared system (remember the central server development paradigm for Unix) a sysadmin would perhaps want to send output of a command or

any errors so that they could be debugged to paper. Not environmentally friendly but sometimes just seeing it in writing solves your issue. Every *device* has a file handle and you can see all the *standards* are located by listing the `/dev` directory.

```
[controller@localhost ~]$ ls -l /dev/std*
lrwxrwxrwx. 1 root root 15 Sep 20 22:48 /dev/stderr -> /proc/self/fd/2
lrwxrwxrwx. 1 root root 15 Sep 20 22:48 /dev/stdin -> /proc/self/fd/0
lrwxrwxrwx. 1 root root 15 Sep 20 22:48 /dev/stdout -> /proc/self/fd/1
[controller@localhost ~]$
```

Figure 6.3: *Standard I/O, ls -l /dev/std**

```
[controller@localhost ~]$ file /proc/self/fd/2
/proc/self/fd/2: symbolic link to '/dev/pts/2'
[controller@localhost ~]$ file /dev/pts/2
/dev/pts/2: character special (136/2)
[controller@localhost ~]$
```

Figure 6.4: *Trail of standard location*

6.4.1 Standard In

The **standard in** is the default method of getting text input into the terminal. Currently this happens to be the keyboard. The **standard in** from the keyboard can be overridden by using the reversed angle bracket `<` to read **standard in** from a file. You can send the content of any file to a command with input redirection only if it accepts input from **standard in** to begin with. Why might you want to send **standard in** from anywhere other than the keyboard? Here are a couple of examples.

```
wc < state-of-the-union-address.txt
mail < complete-works-of-shakespere.txt
```

6.4.2 Standard Out

When Unix was developed the design decision was made that all devices were files. In short, the screen, or teletype, or terminal, is nothing more than a file that is located in the `/dev` directory. By having a **standard out** device this allowed Unix/Linux to exchange standard output devices and not have to modify the output structure of the operating system. Notice too that the outputs are referred to as **tty** which we learned about in chapter 4. When you execute a command, the output is returned by default to **standard out** which in this case is the terminal screen. You can use the `>` and `>>` out redirectors to send the standard output to a file. Be careful, a single `>` will create a file if it does not exist, but will also destroy the content of a previous existing file. A double `>>` will always append, and is non-destructive. When you execute this command you will notice no output comes to the screen. This is because the angle bracket has redirected the output and written it to a file.

```
date > /tmp/todaysdate
cat log.old >> log.new
```

6.4.3 Standard Error

When you type a command that does not execute successfully it returns an error message to the **standard out**. Perhaps the message is *“file or directory not found.”* that is printed to the screen. But that error is not part of the **standard out**, it is part of **standard error**. Standard error can be useful for logging and debugging purposes. If you have a command you want to execute but you want to suppress the **standard error** from showing on the screen, yet log it to a file, you can do so.

6.4.3.1 Suppressing Standard Out and Error

Each of the standards can be referenced numerically. The first element is **__standard in** which has the implied value of **0** or no value at all. The next element, **standard out** has the value of **1**. The final element, **standard error** can be referenced by the number **2**. Note in the script below we are redirecting standard out and error to separate file which can be used for debugging our shell script later on. Standard out and standard error can be redirected together in the bash shell with a single **&** in front of an angle bracket.

```
sudo apt-get -y update 1>/tmp/01.out 2>/tmp/01.err
sudo apt-get -y install nginx 1>/tmp/02.out 2>/tmp/02.err
sudo systemctl nginx start 1>/tmp/03.out 2>/tmp/03.err
```

Exercise: Use the single angle bracket to redirect the output of a command to a file. Technically this command is a “1” followed by an angle bracket “>” but the one is implied and can be left out. What happens when you try this command: `ls -l > ./dir-list.txt; cat ./dir-list.txt`

Exercise: You can suppress the standard error output as well by redirecting it to a file. What happens when you type this? `ls -l /new-top-secret 2> ./error-report.txt; cat ./error-report.txt`

Exercise: Both standard out and standard error can be redirected together. This is useful if you are running a script as a system process or as part of a scheduled task. What happens when you type these commands? `ls -l ~; ls -l /new-top-secret &> error-and-out.txt`

Exercise: As a final convention anytime you want to *throw away* or completely suppress output you can redirect it to `/dev/null`. This directory is called the *bit bucket* or the *black hole* any output directed to it is destroyed irrevocably. It is useful if you just want a command to execute but not display and output or error messages. This is best used in shell script when looking for the existence of a command binary or file. You just want the confirmation that the file exists not any output. What happens when you type the `ls -l /topsecret 2> /dev/null` command?

6.5 History and Usage of Pipes

6.5.1 Douglas McIlroy



Figure 6.5: *Douglas McIlroy*

Douglas McIlroy was the manager of the Bell Labs Computing Techniques Research Department from 1965 to 1986 where Ken Thompson, Dennis Ritchie, and Brian Kernighan worked under his direction. He gave support to Unix and encouraged its development. He even built several Unix commandline binaries such as `spell`, `diff`, `sort`, `join`, `graph`, `speak`, and `tr`². Most importantly McIlroy envisioned the concepts of **pipes** or **pipelines**. This idea allowed

²https://en.wikipedia.org/wiki/Douglas_McIlroy

for the output of one command to be directed as the input of another command—making a pipeline. This was in compliance with Thompson’s idea of small command binaries doing only one thing. So by 1973 McIlroy had convinced Thompson to modify and add this feature to Unix.

“Pipes were first suggested by [M. Doug McIlroy](#), when he was a department head in the Computing Science Research Center at Bell Labs, the research arm of AT&T (American Telephone and Telegraph Company), the former U.S. telecommunications monopoly. McIlroy had been working on macros since the latter part of the 1950s, and he was a ceaseless advocate of linking macros together as a more efficient alternative to series of discrete commands. A macro is a series of commands (or keyboard and mouse actions) that is performed automatically when a certain command is entered or key(s) pressed.

McIlroy’s persistence led Ken Thompson, who developed the original UNIX at Bell Labs in 1969, to rewrite portions of his operating system in 1973 to include pipes. This implementation of pipes was not only extremely useful in itself, but it also made possible a central part of the Unix philosophy, the most basic concept of which is modularity (i.e., a whole that is created from independent, replaceable parts that work together efficiently).³”

[2005 audio presentation by Doug McIlroy about the early history of Unix development](#)

6.5.2 Additional Commands Used for Manipulating Standard Out

The best way to envision pipes is to think of them as exactly what they are called—physical pipes in which water travels through. To maximize your understanding of the concept of *piping* output we will introduce a series of additional command binaries to the list of essential command binaries that was introduced in last chapter and show you combined examples.

who Allows a user to query which accounts are logged into the system at the current time. **Usage example:**

```
who
```

sort Similar to the `cat` command, `sort` will display the content of a file to standard out with the added feature of sorting the content alphabetically. Note that what is sorted is just a copy of the output to the screen. The original file is left unaffected. The `sort` and `uniq` commands are often used together. The `\n` character provides a newline or carriage return **Usage example:**

```
echo -e "orange\npeach\ncherry" > fruits.txt; sort fruits.txt
```

uniq The `uniq` command is used for reporting or filtering out repeated lines in a file. Though `uniq` is a full command binary it is mostly used as a filter through which input has been piped ⁴. For example if we had an webserver error log file with the content below, we would use `uniq` to filter out duplicated lines. The `uniq` command would collapse the first 5 lines into one unique line for output. The `-c` option will add a count after each unique line and the `-d` will only show a line if it has two or more occurrences. **Usage example:**

```
cat error.log
[Sun Mar 16 16:35:16 2014] [error] [client 64.131.110.29] File does not exist:
[Sun Mar 16 16:35:16 2014] [error] [client 64.131.110.29] File does not exist:
[Sun Mar 16 16:35:16 2014] [error] [client 64.131.110.29] File does not exist:
[Sun Mar 16 16:35:30 2014] [error] [client 157.55.33.126] File does not exist:
[Sun Mar 16 17:49:05 2014] [error] [client 216.152.249.242] File does not exist:
[Sun Mar 16 18:05:54 2014] [error] [client 75.148.238.204] Invalid method
```

```
uniq error.log
```

wc The `wc` command stands for and is sometimes pronounces *word count*. It is used for printing out the number of newlines, words, and byte count for a file or for standard in. You can use `wc` as part of a filter to count only specific occurrences of a word. The file `hosts.deny` is a file containing IP addresses of systems attempting to brute force hack a server via SSH. The banned IPs have been placed in the `hosts.deny` file for which the system will deny a TCP connection from any address listed in that file. How many have been added? You can count the lines, words, and bytes by executing the command below or just the number of lines by using the `-l` option. **Usage example:**

³<http://www.linfo.org/pipe.html>

⁴<http://www.computerhope.com/unix/uuniq.htm>


```
wc hosts.deny; tail hosts.deny | wc
```

```
wc -l hosts.deny
```

cut The `cut` command will allow you to *cut* up your output based on a delimiter: space, dash, comma, and so forth. The command will print selected parts of lines from a file to standard output⁵. The `-d` option is the delimiter to *cut* on and the `-f` is the field or column number(s) to retrieve. **Usage example:**

```
cut -d ' ' -f1,2 /etc/mtab
```

```
uname -a | cut -d" " -f1,3,11,12
```

echo This command is used for displaying a line of text. By using the `-e` option you can enable escape sequences to be interpreted. The first example will print out a line of text. The second example uses a for loop to print out a single period and then sleep for 1 second. **Usage example:**

```
echo "hello world!"
```

```
for i in {0..10}; do echo -ne '.'; sleep 1; done
```

tee The `tee` command is used to read from standard input and write to standard output. The command below will sort the output of the `hosts.deny` file and well as save that sorted output to a file name `./sorted.txt`. Then that output will be further passed to a `wc` command. Think of the `tee` command as like a t-shaped pipe that lets water flow down as well as across. **Usage example:**

```
cat hosts.deny | sort | tee ./sorted.txt | wc
```

dmesg The `dmesg` command is a shortcut to display the kernel ring buffer—or in other words the kernel's output messages for debugging. **Usage example:**

```
dmesg
```

tail Capitalizing on the *clever hack* mantra of Richard Stallman, the `tail` command produces the same output as the `cat` command yet does it so by showing the last line 10 lines (by default) of a file fist. Note the Unix humor as `tail` is the opposite of `head`. Also a `cat` has a tail... **Usage example:**

```
tail hosts.deny
```

head The `head` command is used to show the first 10 lines (by default) of a command. It is useful for getting a *peek* at what is in a file if you can remember without having to display the entire contents of a file. **Usage example:**

```
head hosts.deny
```

tac The `tac` command is another *clever hack* in that it does the exact same thing as the `cat` command except that it does it in reverse (cat backwards). **Usage example:**

```
tac hosts.deny
```

6.5.3 Pipe Usage

Pipes can be used to chain as many commands together as necessary. This is one of the three main design principles of Unix.

Exercise: In the previous `cut` example we see the `uname` command which is used to print system information to standard out being piped to the `cut` command and only certain fields of the information being displayed. Try it. Type `uname -a` and then `uname -a | cut -d" " -f1,3,11,12`. What is different about the output?

Exercise: You can also add input redirection to a pipe. `dmesg | tail > system-output.txt` This command will take the `tail` (or last 10 lines) of the `dmesg` kernel output buffer and redirect them to a text file for later analysis. We can also pipe data and redirect as well. `cat deny.hosts | uniq -c > deny-results.txt` Multiple commands can be chained together. Try this command, `cat deny.hosts | uniq -c | sort -nr` and then try it again omitting the `| sort -nr` what is the difference?

⁵<http://www.tldp.org/LDP/abs/html/textproc.html>

Exercise: The chaining process can grow pretty extensive. The `ps` command is used to list system processes that are running and will be covered in detail in a later chapter. This command will look for every running process that has the `system+wildcard` name in it, sort it, pipe that output to a `tee`—which saves that formatted output to a file. Then that sorted text will be passed on as standard in for a `cut` command to filter out columns via spaces and cut the first column and display this text to standard out.

```
ps -ef | grep system* | sort | tee ~/processes.txt | cut -d ' ' -f1
```

Exercise: A variation on the command above but here there is a second `tee` command and a final passing to a `wc` command that will count the occurrences of the `system*` value. `ps -ef | grep system* | sort | tee ~/processes.txt | cut -d ' ' -f1 | tee ~/columns.txt | wc`

6.6 Commands for Finding, Locating, and Pattern Matching

6.6.1 `grep`

The `grep` command is a powerful pattern matching and searching tool for the shell. Originally a feature of the `ed` line editor program - it stood for `g/re/p`—global regular expression print. A `grep` command will search inside of a given text file or directory looking for lines of text that match the pattern you give it. You can use shell meta-characters from above or literal text strings as the searching parameters. There are even more advanced search patterns called *regular expression* which is almost a programming language in itself that allows for complex text queries. We will not cover *regular expression* in depth in this book but talk about it as it relates to helping us complete our jobs.

The `grep` command ⁶ has many options to modify the returned output. Some of the more common ones are listed here ⁷:

- The `-i` option causes a case-insensitive search.
- The `-w` option matches only whole words.
- The `-l` option lists only the files in which matches were found, but not the matching lines.
- The `-r` (recursive) option searches files in the current working directory and all subdirectories below it.
- The `-n` option lists the matching lines, together with line numbers.
- The `-v` (or `-invert-match`) option filters out matches.
- The `-c` (`-count`) option gives a numerical count of matches, rather than actually listing the matches.

Exercise: Commands in which large amounts of text are going to be displayed can be filtered and then piped to a `less` command for viewing. If you view the `hosts.deny` file contents you will see it has two columns of text: first column is the service name, the second is the IP address that is banned. With over 3000+ entries you can use pipes and filters to narrow down the output. For example let us say that you are looking for every line that has an IP that starts with `216.*` and then count those number of occurrences?

```
cat hosts.deny | grep "sshd: 210" | wc
```

```
sshd: 66.109.17.156
sshd: 219.136.240.153
sshd: 212.235.32.75
sshd: 119.68.100.28
sshd: 82.200.130.235
sshd: 124.127.116.130
```

Figure 6.6: *Format of the hosts.deny file*

What would happen if you added a `| less` command to the end like this? `cat hosts.deny | grep "sshd: 210" | less ?`

Can we rewrite the above command to be more efficient? What if we use the `cut` command? `cat hosts.deny | cut -d ' ' -f2 | grep ^210 | less` This will cut out the first column, search for all lines starting with 210 (the `^` in this expression tells `grep` to look only at the beginning of the line), and then pass the results to the `less` command.

⁶<http://www.tldp.org/LDP/abs/html/textproc.html>

⁷<https://www.gnu.org/software/grep/manual/grep.html>

6.6.2 find and locate commands

The `find` command is used to search for files in a directory hierarchy. This command has additional parameters that can be used to find files based on time or size criteria. This is useful searching for files based on when it was created or for finding large files that might be cluttering up the system. The `find` command can also receive shell meta-characters as part of the search pattern. All numeric values can be denoted in this fashion:

Character	Value
+n	for greater than n
-n	for less than n
n	for exactly n

Some of the common categories you can use for `find` are: file time parameters, file size parameters, user ownership parameters, and permission parameters.

6.6.2.1 TIME

-amin n File was last accessed n minutes ago.

-anewer file File was last accessed more recently than file was modified. If file is a symbolic link and the `-H` option or the `-L` option is in effect, the access time of the file it points to is always used.

-atime n File was last accessed n*24 hours ago. When `find` figures out how many 24-hour periods ago the file was last accessed, any fractional part is ignored, so to match `-atime +1`, a file has to have been accessed at least two days ago.

-cmin n File's status was last changed n minutes ago.

-cnewer file File's status was last changed more recently than file was modified. If file is a symbolic link and the `-H` option or the `-L` option is in effect, the status-change time of the file it points to is always used.

-ctime n File's status was last changed n*24 hours ago. See the comments for `-atime` to understand how rounding affects the interpretation of file status change times.

6.6.2.2 SIZE

-size n[cwbkMG] File uses n units of space. The following suffixes can be used:

- b for 512-byte blocks (this is the default if no suffix is used)
- c for bytes
- w for two-byte words
- k for Kilobytes (units of 1024 bytes)
- M for Megabytes (units of 1048576 bytes)
- G for Gigabytes (units of 1073741824 bytes)

6.6.2.3 USER

-user uname File is owned by user uname (numeric user ID allowed).

-perm 000 File is owned by user uname (numeric user ID allowed).

Exercise: `find . -perm 664`

Exercise: `find ~ -mtime 0`

Exercise: `find ~ -size +100M`

Exercise: `find ./ -size +1M`

Exercise: `find ./ -name "*.log"`

Exercise: `find ./ -size +1M`

6.6.2.4 Locate

An alternative to using `find` is the `locate` command. This command is often quicker and can search the entire file system with ease. The `locate` command is not part of the GNU coretools so it may not be present on your Linux distro by default. You can install it by typing `sudo apt-get install mlocate` or `sudo dnf install mlocate`. The `locate` command reads one or more databases prepared by `updatedb` and writes file names matching at least one of the PATTERNS to standard output, one per line ⁸.

There are actually 3 versions of `locate`, GNU `locate`, `slocate`, and `mlocate`. What the `locate` command does is make a database of all files on the system, thereby making lookups faster. You simply run the `sudo updatedb` command to update the index (done every time on reboot).

Example usage: `sudo updatedb; locate *.png; locate chapter-05.md`

Example usage: Compare the output and amount of time this command takes: `sudo find / -name chapter-05.md` and `locate chapter-05.md`—`locate` is clearly faster because databases are good at looking these kind of things up quickly.

6.7 Hidden files and single dot operator

In the previous chapter we talked about the single-dot operator—which is a short cut for *right here* or your present working directory. If the single-dot precedes a file or directory name that file or directory becomes hidden. Not completely invisible but by using the `ls` or `ls -l` commands you will not see these files. You need to add the `-a` option to see all hidden directories. Usually hidden directories are reserved for important user configuration files or system files. It is a convention not a rule, but as you have seen so far, tradition is rule in Unix.

Example usage: Compare the output of these two commands: `ls -l ~` and `ls -la ~`—what new files or directories appear?

Example usage: You can create a hidden file by prefixing a filename or directory with a single dot: `touch .top-secret.txt` or `mkdir ~/.topper-secret`

6.8 Chapter Conclusions and Review

In this chapter we covered and explored the power-tools of the commandline. We learned about shell meta-characters and how they enhance the usability of the commandline. We examined additional command binaries used for manipulation and modification of output to the screen. We learned about the standard input, output, and error devices and how input can be redirected to them via the arrow operators. Finally we covered `find`, `locate`, and `grep` tools for searching in files and for files. This chapter is preparing us to combine of these features into a shell script.

6.8.1 Review Questions

Shell Meta-Characters, Pipes, Search, and Tools

Chapter 06 review questions

1. What is the name for characters that have special meanings in the Linux shell?
 - a. special chars
 - b. marked characters
 - c. shell characters
 - d. shell meta-characters
2. Assume your `pwd` is `~`. If you wanted to list every directory only that started with the letters “Do” what would be the command?
 - a. `ls -l`
 - b. `ls -la [Dd]o`
 - c. `ls -l Do`
 - d. `ls -l Do*`

⁸<http://manpages.ubuntu.com/manpages/utopic/man1/mlocate.1.html>

3. In figure 6.2 in Chapter 06 which of the 3 blue boxes is the step where shell meta-characters are transformed into text?
 - a. Lexical analysis and parse
 - b. Execution
 - c. Builtins
 - d. Expansion
4. Which meta-character allows you to string commands together regardless of the successful execution of the previous command?
 - a. &&
 - b. ;
 - c. \+
 - d. ||
5. Which meta-character allows you to string commands together but will exit if one of the commands fails?
 - a. &&
 - b. ;
 - c. \+
 - d. ||
6. Which meta-character is the wildcard (0 or more matches)?
 - a. ?
 - b. **
 - c. &
 - d. *
7. Which meta-character is the single character wildcard?
 - a. ?
 - b. '
 - c. &
 - d. *
8. Square braces [] indicate sets or _____ of characters to be processed
 - a. numbers
 - b. letters
 - c. characters
 - d. ranges
9. If you wanted to use brace expansion and create a series of nine files named: team1, team2, team3, etc etc all at once—what command would you use?
 - a. create file{1..9}
 - b. file{1..9}
 - c. touch file[1..9]
 - d. touch file{1..9}
10. If you wanted to assign the value of /etc/alternatives/java to a shell variable named JAVA_HOME—what would be the proper syntax?
 - a. JAVA_HOME = /etc/alternatives/java
 - b. /etc/alternatives/java = JAVA_HOME
 - c. JAVA_HOME=/etc/alternatives/java
 - d. \$JAVA_HOME=/etc/alternatives/java
11. What would be the proper syntax to display the content of a variable named JAVA_HOME in the shell?
 - a. echo JAVA_HOME
 - b. cat JAVA_HOME
 - c. print JAVA_HOME

- d. `echo $JAVA_HOME`
12. There are 3 standard I/O devices in a Linux system, standard in, standard out, and _____
- a. standard air
 - b. standard I/O
 - c. standard x
 - d. standard error
13. Standard In is what device by default?
- a. mouse
 - b. screen
 - c. tty
 - d. keyboard
14. Standard Out is what device by default?
- a. mouse
 - b. screen
 - c. X
 - d. keyboard
15. What meta-character can you use to redirect standard out to a file? (Choose all that apply.)
- a. `>`
 - b. `»`
 - c. `<`
 - d. `»>`
16. What meta-character is used to redirect the standard output of one command as the standard input of another command?
- a. `->`
 - b. `&&`
 - c. `||`
 - d. `|`
17. Which command is a shortcut to display the kernel's output messages?
- a. `kern`
 - b. `&kern`
 - c. `top`
 - d. `dmesg`
18. Which command is used to search within files using textual filter patterns?
- a. `find`
 - b. `locate`
 - c. `slocate`
 - d. `grep`
19. Which command can be used to count lines that are in a text file?
- a. `count`
 - b. `wc`
 - c. `lines`
 - d. `uniq`
20. Which command can be used to find unique line occurrences in a text file?
- a. `sort`
 - b. `uniq`
 - c. `wc`
 - d. `who`

6.8.2 Podcast Questions

Listen to the FLOSS podcast number 73 with [Tim O'Reilly](http://twit.tv/floss/73) - <http://twit.tv/floss/73>

- Who is Tim O'Reilly? ~3:00-5:00
- What is OSCON? ~6:45
- Who coined the term Web 2.0? ~13:34
- What did we learn from the IBM PC? ~18:30
- What is Web 2.0? ~19:30
- Open Source vs Open Data - what does Tim O'Reilly think is the ultimate destination for computing? ~23:00
- Where is the money made in open source - software or data? ~ 34:00
- What prediction did Tim O'Reilly make in this podcast (2009) that is now coming true? ~51:32
- radar.oreilly.com What is the lag time from articles on this site to the main stream media? ~55:00

6.8.3 Lab

The objectives of this lab will be to use the shell and understand meta-characters, pipes, search, and tools. The outcome will be that you will be able to successfully use meta-characters for file creation, location, modification, and manipulation. You will successfully master the concept of pipes and redirection as well. Resist the temptation to use the GUI file manager and a web browser. All actions will be done through the shell. You can use either an Ubuntu or a Fedora based OS. **Deliverable:** Take a screenshot demonstrating the required output of each command.

1. Issue the command to clone a copy of the textbook code (if you have already done this in Lab 5 no need to repeat the step). Issue the command to `cd` into the `Linux-Text-Book-Part-I` directory. Type the command that will list every file in this directory that has any number of characters and a `.sh` two character file extension of any name
2. Type the command inside the textbook directory will do a *long listing* of Chapters-02, 04, 06, and 08 only
3. Type the command that will copy the file `Chapter-02/chapter-02.md` into the your home directory, then list the content of your home directory. Use the meta-character needed to execute the proceeding commands only if the previous command is true and place all these commands into one single line
4. In your home directory, using the meta-character, create these two series of files: `ubuntu10.txt - ubuntu15.txt` and `redhat10.txt - redhat15.txt`. Create each series using a single command
5. In your home directory, using the meta-character, issue a command to list only the `ubuntu10.txt - ubuntu15.txt` files
6. In your home directory, using the meta-character, create the directories named: `debian10`, `debian11`, and `debian12` with a single command
7. In your home directory, using the meta-character, delete the directories named `debian10`, `debian11`, and `debian12` with a single command
8. In your home directory redirect the output of the `date` command into a file named: **today.txt**
9. In your home directory append the output of the `date +%m%d%Y` command to the file **today.txt** and display the content - you should see two formatted date entries
10. Create a shell variable named `UT`, assign the contents of the command `uptime` to `UT` and print a string to the screen with its value and with a string stating, "The system has been up for:" and then the value of `UT`.
11. Using an Ubuntu Desktop system, execute the following commands: `sudo apt-get -y update 1>/tmp/01.out 2>/tmp/01.err sudo apt-get -y install nginx 1>/tmp/02.out 2>/tmp/02.err` and `sudo systemctl start nginx 1>/tmp/03.out 2>/tmp/03.err`. Issue the command to list the contents of the `/tmp` directory.
12. Display the contents of the `*.out` files using a meta-character in one command and pipe the output to the `less` command
13. Display the contents of the `*.err` files using a meta-character in one command and pipe its output to the `less` command.
14. Type the command `ls -l /topsecret` and redirect both standard out and standard error to a file named: `/tmp/out-and-error.txt`
15. You will find a file named `hosts.deny` located in the directory `files > Chapter-06` of the download of the textbook. It contains a list of IP addresses - what command would you use to count ONLY the number of lines in the file?
16. Using the `error.log` file located in the directory `files > Chapter-06` - what command would you use to count only unique lines and to display their count and only if there is more than 1 occurrence?

17. What command would let you display the content of the `hosts.deny` file, cut out the the second column and sort it?
18. What command would let you search the file `error.log` for the lines that contain the term `robots.txt`?
19. What command would let you count the number of lines that have the term “robots.txt” in the file `error.log`?
20. Using the `hosts.deny` file, what command would you type to display the last 10 lines of the file, cut out the field with the IP address and sort them in ascending order?
21. Use the `grep` tool to search the file `error.log` for the line “Invalid method in request” and print to the screen the lines found.
22. Using tools discussed in this chapter and based on the contents of the file `error.log`, type the commands needed to find the following information: How many unique IPs did the error messages come from?
23. Home many unique URLs based errors (last column), and list all of the unique type of errors (second to last column).
24. Using the `find` command and starting from the `~` directory what would be the command to find all files with the name `.md`?
25. Using the `find` command and starting from your home directory, what would be the command to find all the files that have been modified in the previous hour?

6.8.3.1 Footnotes

Chapter 7

Introduction to Linux Editors, Shell Scripts, and User Profiles

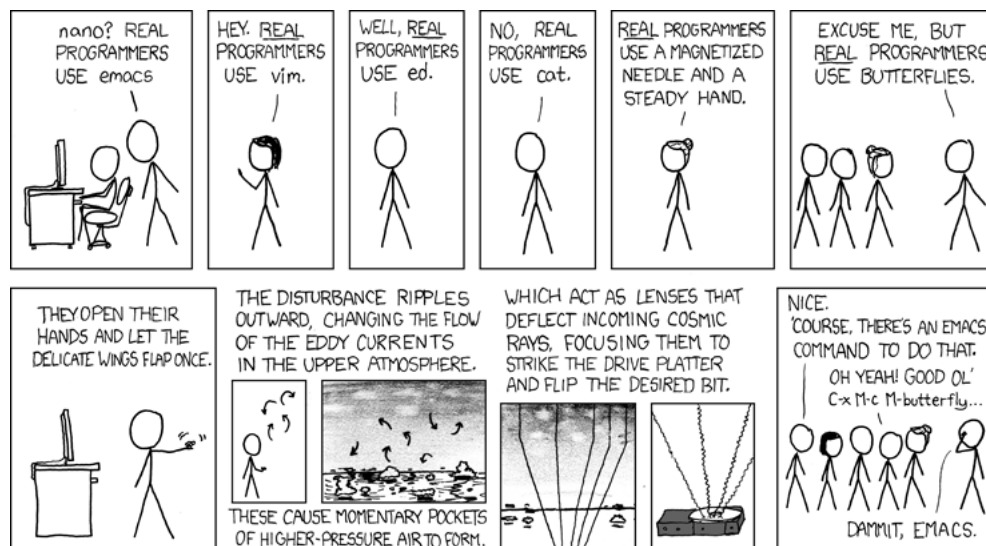


Figure 7.1: *Real programmers...*

7.1 Objectives

In this chapter we will be continuing our exploration of the Linux Shell. We will be introducing editors and examining their use in managing our Linux system. We will also look at understanding user environments and write our initial shell scripts.

- Understand the difference between stream editors and text editors
- Understand and learn how to use the vi(m) editor
- Understand how to use shell scripts to automate tasks
- Understand how to use the system PATH and modify a users profile

7.2 Outcomes

At the outcome of this chapter a user will be able to use the vi editor for creating and manipulating text files and shell scripts. This will give you mastery over the data on your system. You will be comfortable creating a shell script to automate system administration tasks and you will understand how a user's system PATH and profile is modified

and loaded. This will open up the remaining chapters where we introduce additional complexity in writing shell scripts and enable greater system administration via shell scripts and using the vi editor.

7.3 History of Unix/Linux Editors

In the previous chapter we continued learning about essential and additional command binaries that can be executed in the shell. We learned about meta-characters which help expand our abilities to execute repetitive tasks and to simplify searching and file creation. The next concept to introduce is **shell scripts**. Shell scripts are a combination of all of the above features that can be placed into a single text file and run on demand or as a scheduled task. This allows you to have prepared **shell scripts** that can be copied from system to system allowing administrators to build up a *tool belt* of scripts that help them get common tasks done. With that in mind, we come to the concept of the **editor**. In Unix/Linux due to the history and different types of standard in and standard out, there are also different types of editors. The two categories are **stream editors** and **screen editors**. Overtime the distinction has blurred but it is safe to think in these two different categories mostly because each group retains artifacts from its development history.

7.3.1 Stream Editors

The reason we call them **stream editors** is that at their time of development, the modern day 30 inch screen we have, and even X for that matter, didn't exist. So editing was done not via text editors or word processors, but it was done by editing single lines at a time. One line would follow another line - hence a *stream*. The two main editors represented in this category are **emacs** and **vi**. Each of these has evolved a distinct following. The **vi editor** (pronounced *vee-eye*) currently is a Unix Posix standard and the only editor that you will find installed by default on all Unix and Linux distros guaranteed, though some newer Linux distros are changing **vi** out for **nano**. So learning **vi** gives you the ability create shell scripts on any Unix/Linux system.

7.3.2 Emacs

Emacs was originated in 1976 from the AI Labs at MIT, the same place Richard Stallman came from. GNU Emacs was released in 1984 and developed entirely by Richard Stallman himself. In 1980 James Gosling (father of the Java Language) had created his own Emacs in the spirit of opensource called gmacs, but sold his project to a company who re-licensed it with a proprietary license. Emacs is basically a [Lisp language](#) interpreter focusing on macros (or key combinations) to make repeatable actions. Emacs is a very powerful editor (see the cartoon at the beginning of the chapter) and has plugins for email and other functions to exist entirely inside of emacs. In the course of this book we will not be focusing on emacs but that is not because of any deficiency, I recommend you to try it out at least once.

In its normal editing mode, GNU Emacs behaves like other text editors and allows the user to insert characters with the corresponding keys and to move the editing point with the arrow keys. Escape key sequences or pressing the control key and/or the meta key, alt key or super keys in conjunction with a regular key produces modified keystrokes that invoke functions from the Emacs Lisp environment. Commands such as save-buffer and save-buffers-kill-emacs combine multiple modified keystrokes¹. GNU Emacs is an extensible, customizable text editor—and more. At its core is an interpreter for Emacs Lisp, a dialect of the Lisp programming language with extensions to support text editing^{2 3}.

7.3.3 The vi Editor

The other stream editor is the **vi editor** or just **vi** (pronounced *vee-eye*). The creator of the **vi editor** was [Bill Joy](#) at UC Berkeley. His intent was to extend the original ideas behind Ken Thompson's editor which was named *ed*. The **vi** editor is written in the C language but you don't code it in C, Unlike emacs which exposes its LISP interpreter to the user. The history of **vi** varies widely from that of Emacs because **vi** is not a GNU project. The chart below shows the history of the **vi editor**.

¹GNU Emacs

²<http://www.gnu.org/software/emacs/>

³Emacs Dired buffers by Emacs development team - Ferk (user who took this screenshot). Licensed under CC BY-SA 3.0 via Commons.

Editor	Year Released	Originator
ed	1971	Ken Thompson (original Unix)
em	1976	George Coulouris
ex	1978	Bill Joy, Charles Haley, included in BSD
vi	1979	Bill Joy
vim	1991	Bram Moolenaar

Initially Ken Thompson’s editor worked well for what he needed. But the commands were very cryptic and made using the Thompson editor very difficult. Ken Thompson’s original shell *ed* is still available for [download on Ubuntu and Fedora](#) if you are interested to see what it was like to use Unix back in the early 70’s

Sometime in 1976 an AT&T co-worker, George Coulouris, while working on sabbatical at Queen Mary’s College London, extended Thompson’s editor and added some usability features. Continuing the clever hack, he named the editor **em** meaning *ed for mortals*. Editors at this time were designed for display terminals that did not have dedicated ram (expensive at the time). The editors only modified a line at a time and were called [Line Editors](#). Because of slow screens and the high price of memory you had the choice of using line editors or displaying the content of a file. Not until the 1980’s did the concept of visual editing really catch on as technology made it possible.

In 1978/89, out at Berkeley, Bill Joy came into the picture. He helped design an improved **em** called **ex**, *em extended*. This introduced a new visual mode in edition to the line editor features that everyone was used to. This extension to **ex** was called **visual mode** or **vi**. After one year and the changes in technology **ex** shifted from being a *line editor* to a *visual editor* primarily. Hence in 1979 by the time of the second BSD Unix release, **ex** was hard linked to permanently launch in **vi** mode. Thus, vi is not really the evolution of ex, vi is ex ⁴.

7.3.4 Relationship of vi and vim

In January of 1983 AT&T’s UNIX System V adopted **vi** as their standard editor. This put **vi** in the hands of everyone using commercial Unix from AT&T as well as anyone using BSD Unix—which up to that point meant almost everyone in the commercial world. But it was not until June of 1987 that [Stevie](#) (ST editor for VI enthusiasts), a limited vi clone appeared. In early January, 1990, Steve Kirkendall posted a new clone of vi, Elvis, to the Usenet newsgroup comp.os.minix, aiming for a more complete and more faithful clone of vi than Stevie. It quickly attracted considerable interest in a number of enthusiast communities⁵. Andrew Tanenbaum quickly asked the community to decide one of these two editors to be the vi clone in Minix;⁶ Elvis was chosen, and remains the vi clone for Minix today.

But at UC Berkeley, Keith Bostic wanted a “bug for bug compatible” replacement for Joy’s vi for BSD 4.4 Lite. The original **vi** code was encumbered by AT&T licensing because Bill Joy had extended Thompson’s original code which technically belonged to AT&T. Using Kirkendall’s Elvis (version 1.8) as a starting point, Bostic created [nvi](#), releasing it in Spring of 1994.⁷ FreeBSD and NetBSD continue to use nvi to this day.

In 1991, Bram Moolenaar created a port of **vi** called **vim** *vi improved*. Vim was created under a GPL compatible free license and it is compatible with the large majority of **vi** functionality and extends to add some modern features like unlimited undo/redo for example. Because of this **vim** is available for all Linux based systems, BSD, Windows, and others. Some distros link **vi** to **vim** replacing it out right.

7.3.5 vi has a Sharp Learning Curve

Many people will say that the **vi editor** has a sharp learning curve and not to use it. I believe that is a spurious argument. The **vi editor** is not a text editor comparable to notepad, but vi was a specific tool developed to create complex interaction with text in as few key strokes as possible. Learning to play the guitar is difficult in the beginning but once you have the muscle memory to do it you can become an expert player that can make beautiful music that few others can. The power of **vi editor** is in the ability to do line editing and visual editing all from the **vi editor**, the ability to search and find, execute internal commands, even use grep and regex for complex pattern matching and replacement from within vi. Keeping your fingers on the keyboard constantly moving keeps your fingers and mind occupied. Nothing takes more time than to change “contexts”. Don’t abandon it because it is hard! You will

⁴<https://en.wikipedia.org/wiki/Vi#Distribution>

⁵https://en.m.wikipedia.org/wiki/Vi#Ports_and_clones

⁶[https://en.wikipedia.org/wiki/Elvis_\(text_editor\)](https://en.wikipedia.org/wiki/Elvis_(text_editor))

⁷https://en.wikipedia.org/wiki/Vi#Ports_and_clones

eventually be working on systems that have no GUI at all: FreeBSD or Ubuntu Server or RHEL or CentOS you will have to use vi.

7.3.5.1 vi and vim

By using vim as a text editor we can create shell scripts which are collections of shell commands with meta-characters and some control logic. Ubuntu links to vim directly as seen in the image below. Fedora keeps two distinct binaries **vi** and **vim** but both of them link back to **vim**. You can use the `which vi` and/or `which vim` to find out the default state of vim on your distro.

7.3.6 The 3 vi Modes

Example Usage: Let's invoke vim from the shell. Open up a terminal and type `vim notes.txt`, what happens? You see a screen like this. All those tilde marks (~) mean that those lines don't exist—they are visual place holders. If you receive an error that vim is not installed you will need to install it via your package manager.



Figure 7.2: vi initial screen

The **vi editor** has 3 modes:

- 1) COMMAND mode used to position the cursor
- 2) INSERT mode used to insert/delete text
- 3) EX mode used to issue commands that edit lines and change the display of the vi editor.

To transition from command mode to insert mode you use the **ESC** key. Hitting escape plus one of the text modification commands will automatically take you to **INSERT** mode. You will know you are in INSERT mode because the bottom of the screen will say INSERT.

Example usage: Let's type a hello world message in the vi editor. Continuing from the example above, to be able to insert text to the file you need to switch modes to INSERT mode. Hit **ESC** then **i** and then type `hello world!`

Example usage: What happens when you hit an arrow key after typing `hello world!`? Why is this? Remember we need to switch modes between COMMAND mode and INSERT command.

7.4 vi Command Cheat Sheet

There are over 150 distinct commands in vi. But to be proficient you need to memorize *initially* about ~25 key commands. I have provided those in the charts below. Some of the commands automatically trigger INSERT mode after you execute them. For instance the **ESC then a** command will append or add text after the end of the current line. It makes sense that you would want to enter INSERT mode after typing an append command. Remember to see the true advantage try to keep your fingers on the *home row* of the keyboard and case matters!

Table 7.2: Positional Commands That Trigger Insert Mode

Command	Command Description
a	add(append) text to the right of the cursor

Command	Command Description
A	add(append) text to the left of the cursor
i	insert text to the left of the cursor
I	insert text at the beginning of the line of text
o	add a new blank line below the cursor
O	add a new blank line above the cursor

Table 7.3: Line Position Commands That Trigger Insert Mode

Command	Command Description
w	moves the cursor to the beginning of the next word
W	same as above but space delimited
b	moves the cursor to the beginning of the previous word
B	same as above but space delimited
0	moves the cursor to the beginning of the current line
\$	moves the cursor to the end of the current line
H	moves the cursor to the upper left corner
M	moves the cursor to the middle left position
L	moves the cursor to the lower left position

Table 7.4: Text Modification Commands That Do Not Trigger Insert Mode

Command	Command Description
x	delete character
X	delete character before the cursor position
dd	deletes the line of the current cursor position
yy	yanks or copies the current line to the clipboard
p	pastes the current line or lines that are in the clipboard
.	repeats the previous command executed
u	undo previous command (unlimited)
Ctrl + r	redo previous command (unlimited)
Ctrl + g	file info
G	go to last line of file
ZZ	shortcut to save the current file and quit out of vi

Example usage: Remember the previous example where we inserted text? Now let's insert a new line of text. How would we do it based the tables above? We need to switch from INSERT mode back to COMMAND mode. This time we type ESC then o to insert a newline below our cursor.

```
I typed ESC and then the letter "i" to put vi in INSERT mode
Could this be the contents of the ever present topsecret file?
~
~
~
~
-- INSERT --                                     3,1                                     All
```

Figure 7.3: *vi insert*

Example usage: What is the command sequence to delete a single character? You would switch to COMMAND mode by typing ESC then x.

Example usage: What is the command sequence to delete an entire line? You would switch to COMMAND mode by typing `ESC` then `dd`.

Example usage: What command sequence would you use to move the cursor position to the end of the current line? You would switch to COMMAND mode by typing `ESC` then `'$'`

7.4.1 vi/ex Mode

In addition to COMMAND mode and INSERT mode there is also one other mode. This is called **EX** mode. This mode is the original **EX** editor. By hitting the `ESC` key and then the colon `:` you will now have the ability to enter additional commands not directly available in the other modes. There are a series of commands you can execute while in **EX** mode. The most important is how to save and how to quit.

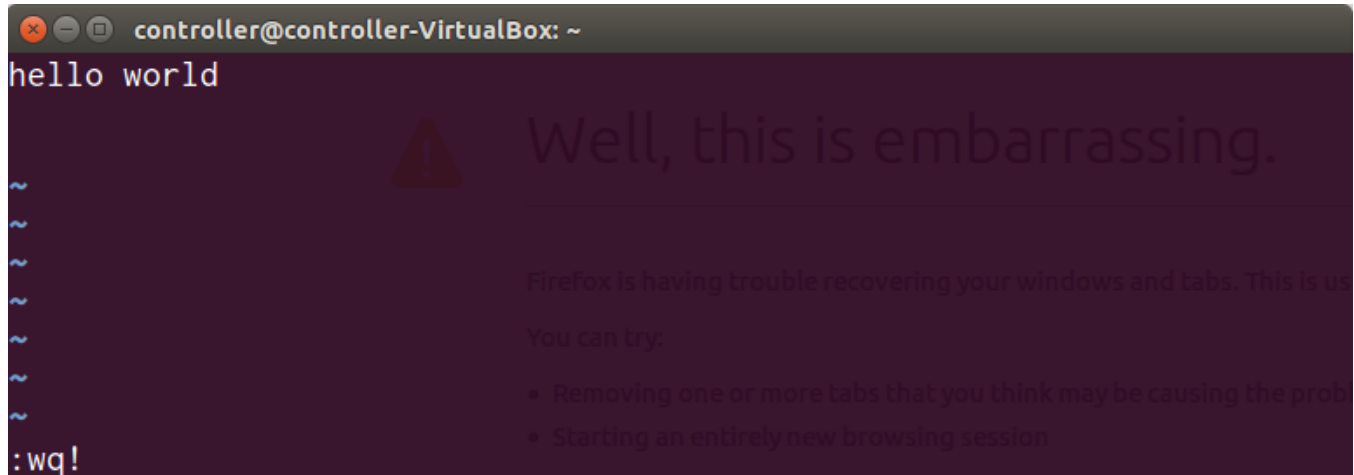


Figure 7.4: *vi ex mode*

Table 7.5: Save and Quit Commands

Command	Command Description
<code>:w</code>	write the contents of a file (save)
<code>:q!</code>	force quit out of the vi editor ignoring any changes
<code>:wq</code>	write and quit a file in vi
<code>:wq!</code>	force write and quit a file in vi

Table 7.6: Additional Meta-Commands

Command	Command Description
<code>:set nu</code>	make line numbers appear in vi
<code>:set nonu</code>	turn off line numbering in vi
<code>:5</code>	move the cursor to a specific line number
<code>:\$</code>	move the cursor to the end of the file

7.4.1.1 Search Forward

EX mode also contains the ability to search for occurrences of text patterns within a text file while using **vi**. By typing the combo `ESC` then `/` you see a slash at the bottom of line of the **vi** editor. This allows you to search for a pattern from your current cursor position. You can type `ESC` then `?` to search the same pattern going up. Hitting the letter `n` will move you to the next result which could be multiple lines away in a large document. You can also use shell-metacharacters to search. The next examples can be recreated using the log file located in the Chapter-07 directory of the included code under the *files* directory. This example uses the file named `u_ex150911.log` which is

an IIS webserver log for a WordPress Installation. Each of these commands needs to be prefaced by an ESC push to change modes.

```

2015-09-12 03:04:22 64.131.110.16 GET /wp-content/uploads/2015/05/centos
15.63.225 Mozilla/4.0+(compatible;+MSIE+7.0;+Windows+NT+6.1;+Trident/4.0
0727;+.NET+CLR+3.5.30729;+.NET+CLR+3.0.30729;+Media+Center+PC+6.0;+InfoP
2;+.NET4.0C;+MSOffice+12) 200 0 0 249
2015-09-12 03:13:20 64.131.110.16 GET / - 80 - 180.76.15.24 Mozilla/5.0+
r/2.0;+http://www.baidu.com/search/spider.html) 200 0 64 702

```

Figure 7.5: *vi search*

`/[Mm]ozilla` will search forwards for any lines containing either *Mozilla* or *mozilla* and highlight each occurrence in the file you are editing in **vi**.

7.4.1.2 Search Backwards

`?Mozilla` will search the file backwards for the word *Mozilla*.

`/Mozilla\?\.0` This is where we can combine shell meta-characters inside of **vi** for searching for Mozilla versions

`?MSIE\\+[6-8]*` This allows for backwards shell meta-character search. In this case notice the introduction of the escape character `\`. Normally the `+` sign has meaning but in our pattern we want to find all the old versions of Internet Explorer 6-8 that are visiting our blog. To do this we pass the line escaping the `+` because we want it to match as a text character not as a shell meta-character.

7.4.2 vi/ex Mode Find and Replace Globally

vi also has the ability to find and replace via a single line or globally. By typing the **ESC** then `:` you will enter the same **ex** mode mentioned above when learning about saving and quitting files. See the sample file `2016-05-31-using-s3cmd.md` located in `files > Chapter-07 > markdown` directory of the files folder.

`:s/Ubuntu/Fedora` The `s` tells us it is a single find and replace or substitute. This is a single instance replacement.

`:s/fall2020/spring2021/g` This command the `s` tells us to substitute the word *fall2020* for the word *spring2021* and the trailing `g` means every occurrence on that line.

`:1,$s///\//g` There is a wildcard option for the line range as well using the `%`

`:%s///\//g` This command has a range prefix, the `1` tells the replacement to start from line 1 and continue to line `$` which is the last line of the file, and replace all occurrences (replace all) of `/` which is the html code for a `/` and note the escapes needed to replace it with a `/`

`:47,86s/<br \>/\//g` This command tells us to do the replacement of lines 47-86 and strip out all the extraneous `
` tags. Note the backslash to escape the forward slash.

7.4.3 Why vi Key Bindings are as They Are

When looking at the patterns of the key-bindings in **vi** they seem a little strange. The reason they were created the way they were had to do with the brand of terminal that **vi** was created on. Remember the standard IBM keyboard we are used to using wasn't created until 1981 on the [IBM PC 5150](#). The type of terminal and keyboard in use at UC Berkeley by Bill Joy was, at that time a competitor to the DEC VT 100 terminals, called the [ADM-3A terminal](#)⁸. It happened that the ESC key was where the modern caps lock key is and that is why ESC is the key used to change modes. The convention just stuck, Unix is more about tradition than reason one could say.

7.4.3.1 A Note About Bill Joy

In some ways Bill Joy could be seen as the west coast version of Ken Thompson. Before Stallman left MIT and started GNU Bill Joy was working hard as a graduate student at Berkeley out in California. He played a large part in helping to further develop BSD Unix. Last chapter we mentioned that he created the C shell and he is also the creator of the

⁸KB Terminal ADM3A by No machine-readable author provided. StuartBrady assumed (based on copyright claims). - No machine-readable source provided. Own work assumed (based on copyright claims).. Licensed under CC BY-SA 3.0 via Commons.

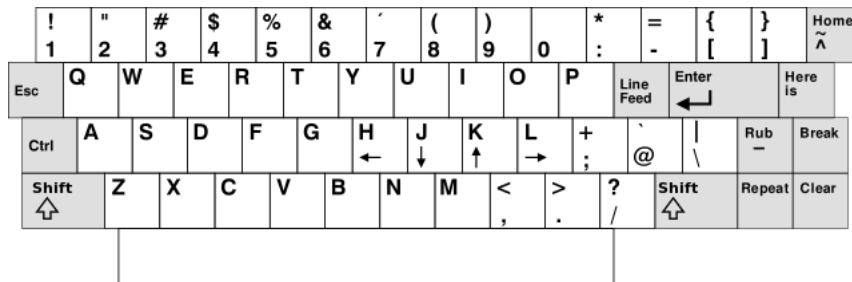


Figure 7.6: *Original ADM-3A Keyboard Layout*



Figure 7.7: *Bill Joy - creator of vi*

vi editor. He left Berkeley in 1982 with 3 other grads from Stanford to form Sun Microsystems, which would play a large role in the commercial Unix world and innovated many technologies (Java). Joy stayed on at Sun until 2003.

In the year 2000 Bill Joy wrote a seminal paper called, “[The Future Doesn’t Need Us](#)”. In the paper he was shocked by the speed of the progress scientific futurist community lead by [Ray Kurzweil](#) coupled with how little they were examining ethics in the face of technological challenges. Ironically Ray Kurzweil is currently employed by Google, whose corporate motto until recently was *Don’t be evil*. Bill Joy said in quote,

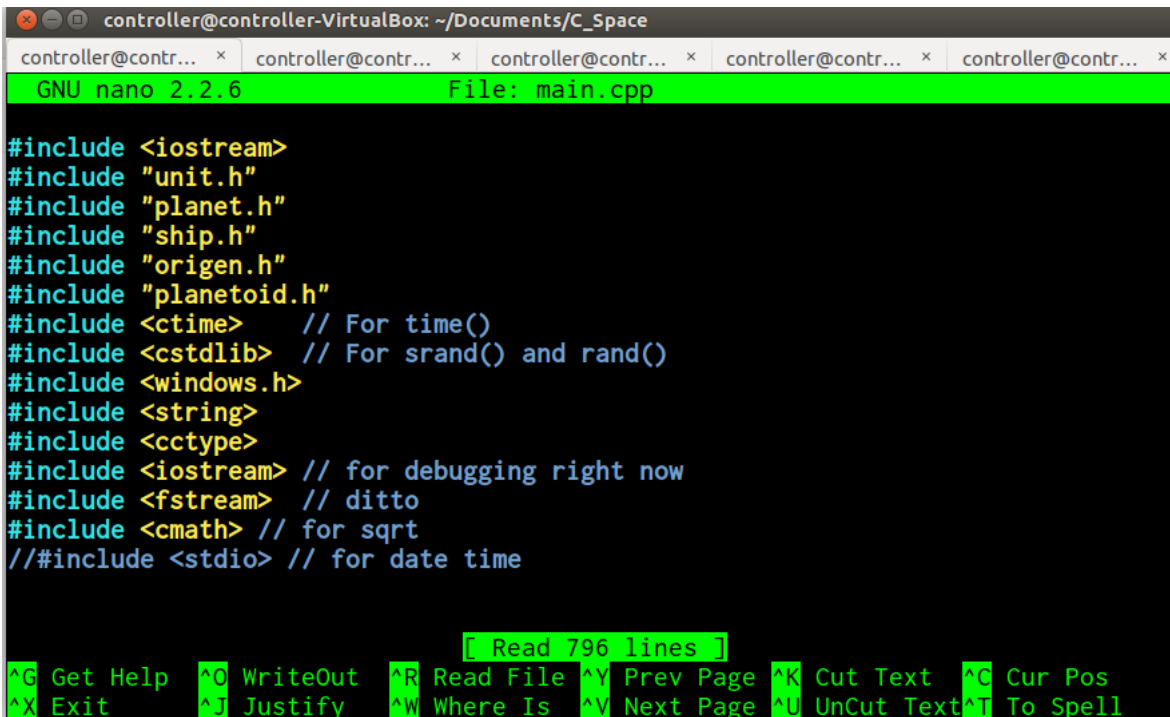
“From the moment I became involved in the creation of new technologies, their ethical dimensions have concerned me, but it was only in the autumn of 1998 that I became anxiously aware of how great are the dangers facing us in the 21st century. I can date the onset of my unease to the day I met Ray Kurzweil, the deservedly famous inventor of the first reading machine for the blind and many other amazing things.”

7.4.4 Screen Editors

The second family of editors differs from the first in that they were created after X was fully implemented and are fully screen oriented, menu driven and have no concepts of what vi and emacs do in the way of line editor functions. There is a second sub-category of screen editors called GUI editors.

7.4.4.1 GNU Nano

[GNU Nano](#) was created in 2000 as a GPL replacement for a common non-free text editor that had come from the University of Washington called PINE. The design is simpler than vim and has become a popular alternative. Nano relies on using the *Control* key in combination with other keys for action. For example `^O` to save and `^X` to quit a file—these commands are listed at the bottom of the screen. Unlike vim, there are no modes, so you are always in *insert* mode and can use the arrow keys and type as if you were in a regular GUI based text editor. For Fedora 33, GNU Nano is to replace vim as the default editor. Nano is very similar to editors such as notepad but has features similar to vim and VSCode. Nano is entirely rendered in text.



```
controller@controller-VirtualBox: ~/Documents/C_Space
GNU nano 2.2.6 File: main.cpp
#include <iostream>
#include "unit.h"
#include "planet.h"
#include "ship.h"
#include "origen.h"
#include "planetoid.h"
#include <ctime> // For time()
#include <cstdlib> // For srand() and rand()
#include <windows.h>
#include <string>
#include <cctype>
#include <iostream> // for debugging right now
#include <fstream> // ditto
#include <cmath> // for sqrt
//#include <stdio> // for date time

[ Read 796 lines ]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

Figure 7.8: GNU Nano

GNU Nano was derived from two UNIX utilities originally used to edit/read/send email: PINE and PICO. These were designed before GNU and Linux and did not have Free Software Licenses. In 2001, GNU Nano was developed to give a free mail reader to the world. The function of Nano changed as GUIs developed and became an regular text or file editor.

7.4.5 GUI Text Editors

7.4.5.1 gedit

The [gedit](#) program was released in 1999 -shortly before the GNOME desktop was released. It is a full fledged text editor with plugin support and syntax highlighting. It is currently part of the GNOME core applications and you will find it installed anywhere GNOME3 is installed.

7.4.5.2 Visual Studio Code

[Visual Studio Code](#) is a new comer to this field. It is a text editor that has built in support for Git, plugins, and syntax highlighting. The advantage is that you can download VS Code for Windows, Mac, and Linux as the project is opensource.

7.4.5.3 Atom

[Atom](#) is a hackable text editor for the 21st century, built on Electron, and based on everything we love about our favorite editors. We designed it to be deeply customizable, but still approachable using the default configuration.

7.4.5.4 Leafpad and Mousepad

- [Leafpad](#) is an opensource notepad clone released in 2004. Leafpad focuses on being light and having minimal dependencies. It provides syntax highlighting and is the default editor for [LXQT](#) and was for Xfce until its replacement by Mousepad. Leafpad is built using GTK+.
- [Mousepad](#) comes standard in Xfce as a notepad clone. It is built using GTK3+.

7.4.5.5 Sublime

[Sublime](#) is the missing editor for the Mac platform. It is free to download but requires a license to be purchased. It is a great product and if you will be doing any significant coding on the Mac platform this is really your only choice. It is available on Windows and Linux as well.

7.4.5.6 Kate

[Kate](#) is the standard editor for KDE based desktops. The KDE equivalent of gedit but more than gedit–Kate can be used as components of larger applications not unlike Emacs. You can install this on any Linux based distro but it requires that KDE components be installed too. If you are using Fedora or Ubuntu this makes the download and install much larger and may add files you don't necessarily want to your system. If you desire to use Kate you may want to look at installing Kubuntu-desktop or finding a KDE based Fedora spin.

7.5 Creating Shell Scripts

Let's open up a terminal and create a shell script. To do this type: `vi list-ip.sh` and from here you will see the screenshot we saw earlier–blank. The first thing we need to type is a shell directive or more commonly called, **she-bang**. Although we are using the *bash* shell you can create scripts that can be run with other shells. The first line of a shell script overrides the default shell and runs the script with the shell you determine.

The first line of any bash script should include: `#!/bin/bash` to make sure that our script is executed with the bash shell. Normally the `#` means a comment, but with the `!` after it followed by a path, the comment function is overruled. `#!` can also be pronounced *crunch bang*.

Not all systems store the bash binary in `/bin/bash`. You will need to check before you hard code that value. You can use the `which` command that will show you the paths to the binaries.

```
controller@controller-VirtualBox:~$ which bash
/bin/bash
controller@controller-VirtualBox:~$ █
```

Figure 7.9: *which bash on Ubuntu*

```
[controller@localhost ~]$ which bash
/usr/bin/bash
[controller@localhost ~]$
```

Figure 7.10: *which bash on Fedora*

Remember the command to insert a new line? That would be `ESC shift + o`. The rest of creating a shell script is the same as you have been doing on the command line. The shell script will execute lines in sequential order allowing you to chain commands together. Let's type some commands to display the last 10 lines of the `hosts.deny` file provided from chapter 6 and add in a message to the user.

```
#!/usr/bin/bash

echo "Here is the content of the ~/Documents/hosts.deny file"
echo "*****"
tail ~/Linux-text-book-part-1/files/Chapter-07/logs/hosts.deny
echo "*****"
```

Now we need to save the file (`w`) and quit out of `vi` (`q`). You can move to `ex` mode by hitting `ESC :wq` to save and quit. Now let us run our shell script. Type `list-ip.sh` on the command line. What happens? Why?

```
controller@controller-VirtualBox:~$ list-ip.sh
list-ip.sh: command not found
controller@controller-VirtualBox:~$
```

Figure 7.11: *Command not found*

7.5.1 System Path

The file is correctly named but we have a problem. The system only knows about command binaries in certain locations. It doesn't know about our user created binaries. How does the operating system know where to look? Simple, type `echo $PATH` and what do you see?

```
# Show the system path from the commandline
echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```

There is a system variable named `$PATH` that is constructed upon boot. It includes the default locations that the essential command binaries, additional command binaries, and user install binaries are located. Every time you execute a command, the system parses the command name and looks down this path to try to find the corresponding binary. Note the absolute paths are chained together with colons `:`. When the shell parser finds the first occurrence—it passes that location and executes that matching binary name. In our case the shell script `list-ip.sh` is located in `~/Documents` which is not in the system path listed in the image above. So how can we reference it? Remember the single-dot operator `./`—that tells the operating system to look here—overriding the system path. Try and type `./list-ip.sh` what happens now?

```
# Executing the command ls /root on the commandline
# controller@controller-VirtualBox:~$ ls /root
# ls: cannot open directory '/root': Permission denied
```

7.5.2 Changing Permissions for Execution

The error message tells us that we have `permission denied`. Remember back to chapter 6 when we dealt with file permissions? In order for a shell script to be executable we need to give it `execute` permission.

We can change permissions by using the `chmod` command. What is the current numeric value of the permissions for the file `list-ip.sh`? What would we need to change it to at a minimum? If you said `764` you would be correct? Why? The minimum we need to do is add the `execute` permission to the owner's permission section. We would go from

```

controller@controller-VirtualBox:~/Documents$ ls -ll list-ip.sh
-rwxrwxr-x 1 controller controller 74 Jul 10 16:09 list-ip.sh
controller@controller-VirtualBox:~/Documents$

```

Figure 7.12: `ls -l list-ip.sh`

this `rw-rw-r--` to this `rwxrw-r--`. We could do that by typing `chmod 765 list-ip.sh`. There is an easier way with group and letter shortcuts.

Table 7.7: Permission Shortcuts

Group	Description
u	Owner of the file
g	Group owner of the file
o	Other (everyone else)

Table 7.8: Permission can be added or negated and combined

Group	Description
<code>chmod u+x list-ip.sh</code>	Gives the owner of the file execute permission
<code>chmod g+x list-ip.sh</code>	Gives the group owner of the file execute permission
<code>chmod o+x list-ip.sh</code>	Gives the other group execute permission (everyone else)
<code>chmod u-x list-ip.sh</code>	Removes file execute permission from the group owner
<code>chmod o-wx list-ip.sh</code>	Removes write and file execute permission from other
<code>chmod ug+rwx list-ip.sh</code>	Owner and group are give rwx permissions together.

You will notice that in the terminal (where supported) files marked executable will turn green. If you use the `ls -lF` flag you will also see that executable files will be marked with an asterisk. Now you can finally execute your command `./list-ip.sh` and see the last ten lines of output from the shell script.

```

controller@controller-VirtualBox:~/Documents$ ls -lF
total 4
-rwxrwxr-x 1 controller controller 74 Jul 10 16:09 list-ip.sh*
controller@controller-VirtualBox:~/Documents$

```

Figure 7.13: *Execute Permission enabled turns green*

7.6 Understanding `.bashrc`

When your system first boots up how does it know how to define system environment variables and system PATH variables? Part of the boot process is to read and source the `/etc/profile` file. This is a system wide profile that all users accounts will inherit from this file.

Next there is local login profile. This is additional customization added to your account after you log in via username/password. There could be any number of files from this list in this order: `~/.bash_profile` or `~/.bash_login` or `~/.profile`. The `.profile` is a hold over from the Korn shell—since bash is ksh compatible.

Then once logged in upon launching a terminal there is another set of profiles to be processed. The `~/.bashrc` file is processed. The template for this file is generally located in `/etc/bashrc` if you want to customize or replace a `~/.bashrc` file. There is one final file that you can use to modify a system environment upon logout and that is the `~/.bash_logout` file.

If you wanted to make modification to your `$PATH` variable to include a directory for the newly made shell scripts you can modify the PATH directly on the command line. The problem is that this variable will only stay in memory for the duration of the terminal window—as soon as it is closed the PATH reverts to the one it started with. In order to permanently modify the PATH we need to modify it in one of the profile files. The best place to make user specific changes is in the `~/.bashrc` file from the commandline. 139

Let's try to add to the PATH. When updating a shell variable we need to use the `export` command so that the system is aware of the new variable value. Think of it as a *refresh* command. In addition to the PATH variable to see

```
# Extending the system path
# Manually added by the user
export PATH=$PATH:/usr/local/bin
```

Example Usage: Type `mkdir ~/Documents/scripts`. Now copy your `list-ip.sh` to this directory. Let's add this directory to our PATH in our `~/ .bashrc` file. Finally before we edit let's print out the content of the PATH system variable so we can see our changes later. How would you do that? Let us open our `~/ .bashrc` file in vi. Now move the cursor position to the bottom of the file. Type `ESC shift+o` to insert a new line. Now type `ESC i` to change to INSERT mode. Type the line `PATH=$PATH:~/Documents/scripts` followed by a new line. Vim cheats and will accept the ENTER key in addition to `ESC SHIFT+o`. Let's export the new variable content by typing `export PATH` now exit vi.

Example Usage: To make the changes we just made register we can do two things. We can reboot the system so all the profiles are re-read but that is a little drastic. A shortcut to re-read and process just the `~/ .bashrc` file is to add a single-dot separated by a space. Type `. ~/ .bashrc` and then display the content of the PATH variable: `echo $PATH` and you should see your new addition appended to the PATH variable permanently.

7.7 Chapter Conclusions and Review

In this chapter we learned about the vi editor and mastered its three modes. We learned how to create a shell script with vi and how to navigate and change modes within vi. We learned about the basic set of vi commands needed to be productive. We learned how to create a shell script, the shortcuts to change the file permissions, and how user profiles are loaded and used.

7.7.1 Review Questions

1. What are the two main representatives of stream editors?
 - a. gedit and kate
 - b. Nano and Joe
 - c. vi and Nano
 - d. vi and Emacs
2. Which family of editors came first?
 - a. Screen editors
 - b. Butterfly editors
 - c. GUI editors
 - d. Stream Editors
3. What type of editor is GNU Nano?
 - a. stream
 - b. text
 - c. A small one
 - d. file
4. Who created the vi editor?
 - a. Richard Stallman, 1984
 - b. Brian Fox, 1989
 - c. Bill Joy, 1979
 - d. Bill Joy, 1983
 - e. Brian Fox, 1979
5. Which of the following sequences of the history of vi is correct?
 - a. Emacs -> ed -> ex -> vi
 - b. ed -> em -> ex -> vi -> vim
 - c. em -> ex -> vi
 - d. em -> ed -> vi -> vim

6. What are the three modes in vi?
7. What is the key you use in vi to transition between COMMAND MODE and INSERT mode?
8. What command sequence (key) in vi will add text to the right of the current cursor position? (just the letter)
9. What command sequence (key) in vi will move you to the beginning of the next word? (just the letter)
10. What command sequence in vi will delete a single line based on the current cursor position? (just the letters)
11. What command sequence in vi will delete 10 lines from the current cursor position? (just the numbers and letters)
12. Which command in ex mode (vi) will save the current file you are working on and exit the vi editor? (include the “:”)
13. In the log file `u_ex150911.log` what would be the ex command to search forward for occurrences of YandexBot? (include the forward slash)
14. Assuming your pwd is `Linux-text-book-part-I` and you have loaded `Chapter-02 > chapter-02.md` into vi, what would be the ex mode command to replace all occurrences of `linux` with `Linux`?
15. Assuming your pwd is `Linux-text-book-part-I` and you have loaded `Chapter-02 > chapter-02.md` into vi, what would be the ex mode command to replace all occurrences of `Linux` with `GNU/Linux`? (remember to escape the `/`)
16. Assuming the your pwd is `Linux-text-book-part-I` and you have loaded `Chapter-02 > chapter-02.md` into vi, what would be the ex mode command to remove all occurrences of the word `Windows`?
17. Assuming a file name `topsecret.sh` has a permission of `644` - what is the shortcut to give just the owner of the file additional permissions to execute the script?
18. Assuming a file named `moretopsecret.sh` has a permission of `757` - what is the shortcut to remove all permissions from the the **other** group?
19. What is the correct command sequence to save or write out a file in GNU Nano?
 - a. `^6`
 - b. `^X`
 - c. `^O`
 - d. `:wq`
20. What is the command to display the contents of the `PATH` system variable on the command line?
 - a. `echo PATH`
 - b. `echo $PATH`
 - c. `echo path`
 - d. `$PATH`

7.7.2 Podcast Questions

View or listen to this Podcast: <https://twit.tv/shows/floss-weekly/episodes/594?autostart=false>

PIP AND THE PYTHON PACKAGE INDEX

- ~7:40 What is pip and how is it related to the Python language?
- ~9:31 What are some of the utilities that a developer who is making a Python package are going to use?
- ~11:15 If someone is getting started with Python what is the right way to install packages?
- ~12:39 Is the number of different package managers and methods of installing Python packages a benefit to the new user?
- ~13:43 Does (did) Pradyun agree with the first speaker?
- ~14:34 Where does the overview of packaging for Python live (URL)?
- ~17:52 Where has Pradyun and pip secured opensource funding from?
- ~19:00 What did Sumana do with the funding received from various companies and organizations?
- ~30:30 How does Sumana explain “DevOps”?
- ~31:37 What happens when you release software more often?

- ~36:10 What is the Hollywood portrayal of those who develop and make software and how is it different from reality of developing software?
- ~39:10 What can a short term Project Maintainer do for a software project?
- ~43:20 How many maintainers did the pip project have in 2017? How many do they have now in 2020? Are they paid developers?
- ~44:44 If pip is one of the most important tool chains in the industry – who funds its development/maintenance?
- ~46:40 What is the programming language that Python is written in?
- ~53:10 What is the tip for what not to do with pip on Linux?
- ~55:15 What is the website (URL) to keep up with Python language announcements?

Book mentioned in the Podcast:

- [The Bug by Ellen Ullman](#)

7.7.3 Lab Chapter 7

Objectives

The objective of this lab is to master vi commands and shell scripts

Outcomes

At the end you will have mastered the basics of vi and now be proficient in the tools of Linux shell scripting

7.7.3.1 Prerequisites

- You will need an additional virtual machine with Ubuntu Server 20.04 installed for this entire lab
- You will need to make sure the vim program is installed
- You will need to make sure the nano program is installed
- You will need to clone the Textbook source code to the Ubuntu Server virtual machine in the home directory
- You will need to install the program vimtutor
 - On Ubuntu by typing `sudo apt-get install vim vim-runtime vim-gtk`
 - On Fedora by typing `sudo dnf install vim vim-enhanced`
 - There is a good text explanation of each of the vim tutor exercises: <https://www.systutorials.com/vim-tutorial-beginners-vimtutor/>

- 1) Using Ubuntu Server, type the command `vimtutor` from the terminal. **Warning:** `vimtutor` requires you to read the instructions carefully!
 - i) This is a 6 part tutorial. You need to follow all the steps of the 6 part tutorial making your changes directly in the file.
 - ii) **Be careful** to save the file to an external location – otherwise IT WILL BE OVERWRITTEN each time you launch the vimtutor command. You can do this by typing `:w ~/Documents/vimtutor.txt` - this way you can edit the file on your local system instead of launching the vimtutor application again. Note you need to use `vim` for this assignment.
 - iii) Take a screenshot as you complete each sub-section (i.e. 2.1 2.2 3.1 4.1)
- 2) From the textbook source code folder: `files/Chapter-07/lab`, copy the file `install-software.shto` your home directory
 - i) Using vim/ex commands, find all occurrences of `python` and replace them with `python3`
 - ii) Save file and quit the vim editor
 - iii) To test your work, give the shell script execute permission and execute it by using `sudo ./install-software.sh`
- 3) From the textbook source code folder: `files/Chapter-07/lab`, copy the file `install-software.shto` your home directory
 - i) Using vim/ex commands, find all occurrences of `python` and replace them with `python3`
 - ii) Save file and quit the vim editor
 - iii) To test your work, give the shell script execute permission and execute it by using `sudo ./install-software.sh`
- 4) In your home directory, using vim, create a shell script named `first-shell.sh` in your home directory that contains the following:
 - i) Add the proper *shebang* on the first line.

- ii) Add two lines of space
 - iii) Store the output of the command `date` into the shell variable named **DT**
 - iv) Add the command that will print out the text: “#####”
 - v) Add the command that will print out the text: “Shell successfully execute at: \$DT”
 - vi) Add the command that will print out the text: “#####”
 - vii) Save the file and quit the vim editor
 - viii) Execute the command to give `first-shell.sh` execute permission
 - ix) Take a screenshot of the output executing `first-shell.sh`
 - x) Take a screenshot of the command used to print the content of the file: `first-shell.sh`
- 5) From the textbook source code folder: `files/Chapter-07/lab`, copy the file `install-software.sh` to your home directory
- i) Open the file `install-software.sh` in GNU Nano
 - ii) On line 5 replace my name with yours - take a before and after screenshot
 - iii) Save the file and quit Nano, then execute the command to show only the **first** 10 lines of the file `install-software.sh`
- 6) Using wither vim or Nano:
- i) Create a shell script named **install-textbook-dependencies.sh** in your home directory.
 - ii) Add the proper *shebang* on the first line, then two lines of space
 - iii) Type the lines at the end of this assignment into your shell script
 - iv) Save the file and exit from your editor
 - v) Give the script execute permission and execute it
 - vi) To test the results, `cd` into the `Linux-Text-Book-Part-1` directory (clone it if you have not) and execute the the script: `./build-linux-and-macos.sh` (the script already have execute permission)
 - vii) To test if the textbook built correctly - `cd` into the directory: **output/pdf**. Issue the `ls` command and you will see two PDF files.

```
# These lines retrieve the pandoc .deb file which is the executable that turns markdown
# into a PDF and ePub
wget https://github.com/jgm/pandoc/releases/download/2.15/pandoc-2.15-1-amd64.deb
sudo dpkg -i pandoc-2.15-1-amd64.deb
```

```
# These three lines are one command. The \ character is a line continuation that
# tells the commandline to treat the next three lines as one single line
sudo apt-get install -y texlive texlive-latex-recommended texlive-latex-extra \
texlive-fonts-recommended texlive-fonts-extra texlive-xetex texlive-font-utils \
librsvg2-bin texlive-science-doc texlive-science
```

```
wget http://packages.sil.org/sil.gpg
sudo apt-key add sil.gpg
sudo apt-add-repository -y "deb http://packages.sil.org/ubuntu/ $(lsb_release -sc) main"
sudo apt-get update
sudo apt-get -y install fonts-sil-charis

sudo apt-get -y install fonts-inconsolata
sudo fc-cache -fv
```

Deliverable:

Submit your GitHub URL for your repo to Blackboard.

7.7.3.2 Footnotes

Chapter 8

Writing Basic Shell Scripts

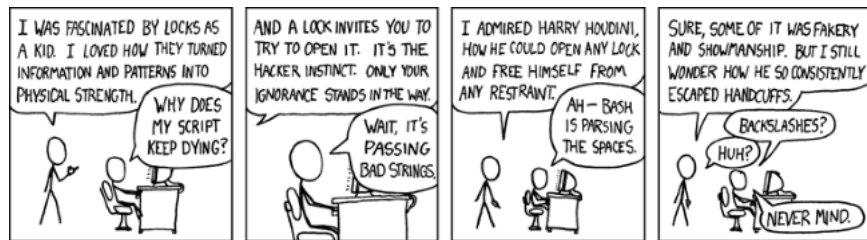


Figure 8.1: *Bash Escape*

8.1 Objectives

This portion of the book begins Part II. The first 7 chapters focused more on the *philosophy* and basic tenets of how Unix and Linux were created. The remaining 8 chapters now will focus on the application of what we have learned and focus on using the opensource technology of Linux. The Objectives of this chapter are as follows:

- Understand how to write and use basic shell scripts
- Understand how to use conditional statements in Bash scripts
- Understand how to declare system environment variables and their scope of existence
- Understand how to use positional parameters as variables into shell scripts
- Understand how to use the scheduling service `cron` for managing shell script automation

8.2 Outcomes

At the conclusion of this chapter you will have furthered your understanding of the `vi` editor and be able to demonstrate how to use control structures in shell scripts. You will also learn about command line variables and how they extend the ability of a shell script to accept dynamic input. You will be able to schedule a shell script to run at a scheduled time by using `cron` service increasing your ability to reduce work by automating repetitive tasks.

8.3 Basic Shell Scripts - Part II

In the previous chapter we were introduced to the simplest of shell scripts. In this chapter we are going to increase the depth of our knowledge.

8.3.1 The Bash Shell

Just like any programming language we cannot have complex logic if we don't have control structures. The two basic ones we want to cover are `if` statements and `for` loops. There are the other traditional control structures but are used less commonly because of the nature of a shell script is a single execution not as a repeated process or system service.

The BASH shell scripting language resembles a traditional programming language. But it is key to remember that it was not designed to be a complete programming language. As you push shell scripts to their limits you begin to see the end of what they are capable of. That is where you see languages like Perl or Python coming in to extend and replace BASH. (Note if you ever find yourself doing serious arithmetic in BASH something is seriously wrong with your design parameters—check again why you are doing this.) BASH is a tool to help automate the repetition of commands.

8.3.2 Shell Script Variables

As we learned previously we can define variables in BASH. These variables are prefixed with a `$` when referenced. In the previous chapter in the `.bashrc` file used to modify the system path, we assigned a new value to the `PATH` variable like this:

```
PATH=$PATH:/home/user/Documents/apps

echo $PATH; echo $path; $PATH=PATH

UT=`uptime`
UT2=$(uptime)

# This variable assignment command will, why?
UT=(`ls -l star[1-3]`)
# This variable assignment will succeed
USA="United States of America"
echo $USA
DIR="monthly-reports-winter-quarter-north-america"
ls $DIR
```

Note that there is no space allowed in variable assignments. `PATH=$PATH` is valid, `PATH = $PATH` will be interpreted in a different way by the shell parser.

Example Usage: Create a shell script with this content below. Save the file, make it executable, and then execute it.

```
#!/bin/bash

# This is a comment - usually by convention variables are in all caps - no spaces, ever!
DT=`date`
# This command outputs the date in a format of our specifications
MDY=`date +%m%d%Y`
echo "*****"
echo "today's date is: $DT"
echo "*****"
echo "Making a folder named MMDDYYYY"
mkdir ~/Documents/$MDY
echo "Finished"
```

In this example we see how the value of `$MDY` is interpreted first and then passed to the argument attached to the `mkdir` command. Note that the `mkdir` command did not have any backticks around it like the command to assign the output of `date` to the variable `DT`. This is because the Shell sees that `mkdir` is a command and begins to interpret the line as such (followed by options and arguments). If you want to encapsulate the output of one command into a shell variable then you need to enclose them in backticks `` ``.

Any variables that are declared have a scope of this script's execution. This means that once the script has finished executing any variables are tossed from memory. If I wanted a variable's name and value to live after the completion of my shell script I can always add the **export** command. The **export** command will take the content of this variable and move it from the memory space of the script's execution and move it into the memory space of the launching shell. This way the variable will live only as long as that terminal session is open—once the window closes those variables disappear from memory because the process that was holding them in a piece of memory is gone too.

Example Usage: Create a shell script with this content below. Save the file, make it executable, and then execute it. Upon completion of that execution, type `echo $DT` what value do you see and why?

```
#!/bin/bash

# This is a comment - usually by convention variables are in all caps - no spaces, ever!
DT=`date`
# This command outputs the date in a format of our specifications
MDY=`date +%m%d%Y`
echo "*****"
echo "today's date is: $DT"
echo "*****"
echo "Making a folder named MDDYYYY"
mkdir ~/Documents/$MDY
export DT
echo "Finished"
```

8.3.2.1 Array Support in Bash

Arrays are a datatype that can be used to associate data into an ordered collection. Arrays in Bash are untyped (all text). There is no support for ArrayLists, maps, queues, or anything of that nature. Bash arrays are used to store related string data. They came about since Bash version 4.0 (2009).

```
# Create a file of names named: names.txt
echo "Joseph" >> names.txt
echo "Evelyn" >> names.txt
echo "Lincoln" >> names.txt

# This will assign the content of each line of names.txt
# to an array named NAMESARRAY
NAMESARRAY=( $(cat names.txt) )

echo ${NAMESARRAY[2]}
# this will print out the second element of the array
# which is Lincoln
```

How can we access these variables? We can make use of some meta-characters that have new special meanings here. First is the *at sign* or `@` which allows us to access all of the elements in an array without having to create a loop. The line below will print out the entire content of the array. The *pound sign* or some people call it a *hash* or *crunch* indicates that we are looking for the length of the array. Note the dollar sign before the element to tell the shell interpreter that this is a variable to be rendered. Also note the the array elements are encapsulated in `{ }`-curly braces to prevent the `[]` square braces from being interpreted as shell meta-characters. As usual elements of an array can be accessed by an index. `echo ${instanceARR[0]}`; `echo ${instanceARR [1]}`; `echo ${instanceARR[2]}`. Remember that arrays are **zero indexed**.

```
# Using the array from the previous example:
echo ${NAMESARRAY[@]}
LENGTH=${#NAMESARRAY[@]}
echo "ARRAY LENGTH IS $LENGTH"
```

8.3.2.2 Positional Parameters

In the case of the command binary `ls -l /etc` the command takes options and arguments. Shell scripts you create have the same ability to accept and parse input from the commandline. Note that this is different from `getopts` which allows you to make a complicated option and argument passing scheme for shell scripts. This is for simple variable parameters to be passed into a script: For example:

```
./delete-directory.sh ~/Documents/text-book Jeremy
```

The content of the shell script is as follows:

```
#!/bin/bash

echo "*****"
echo "The directory to be deleted is: $1"
rm -rf $1
echo "*****"
echo "It was deleted by: $2"
echo $2 > ~/Documents/deletion-log.txt
```

Note that each positional parameter that is passed in to the shell script is simply accessed by a number prefixed by a \$. What do you think would be the value of \$0? You can similarly access the number of variables that are passed into the command line by using the built-in variable: For example:

```
#!/bin/bash
# posparam.sh

echo "This is $0: $0"
echo "This is the length of the number of items passed in $# (not counting $0): $#"
```

```
echo "This is the entire array of items passed in @$@: @$@"
echo "This is the first parameter: $1"
echo "This is the fourth parameter: $4"
```

Upon giving the shell script above execution permission and then executing the line below, what will the three lines output?

```
posparam.sh domo origato Mr. Roboto
```

8.3.3 Control Structures

There are other structures for creating full scale shell scripts that parse user input and create menu like functions. That is beyond the scope of this book.

8.3.4 IF Statements

The structure of the Bash **IF** command is as follows:

```
if TEST-COMMANDS; then CONSEQUENT-COMMANDS; fi
```

The **IF** command starts with a test condition or command. It is followed by a **then** condition which will execute if the test command is *true*. The entire **IF** command is closed by the letters **fi**. Since the scope of BASH is limited compared to a full programming language, **IF** statements are mostly used to test for conditions or the existence of a condition. These common **TEST COMMANDS** are available as built in to BASH in order to accomplish just that. These are called *primaries* and are placed between square braces—space is important ¹. Remember that spaces matter in writing **IF** statements. Unlike C or Java, the Bash interpreter needs those spaces so it can recognize that there is an **IF** statement otherwise it gets confused thinking the characters are just a string. Also the [] needs to have the space otherwise the commandline parser will interpret the square bracket and the number as a shell meta-characters and not delimiters for the test condition.

```
#!/bin/bash
# This is a valid IF statement

if [ -r ~/Documents/final-exam-answers.txt ]
then
    cat ~/Documents/final-exam-answers.txt
else
    echo "File is not readable."
fi

#!/bin/bash
# This is NOT a valid IF statement because the spaces are improper
```

¹http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_01.html

```

if [-r ~/Documents/final-exam-answers.txt]
then
  cat ~/Documents/final-exam-answers.txt
else
  echo "File is not readable."
fi

```

Table 8.1: Primary expressions

Primary	Meaning
[-a FILE]	True if FILE exists.
[-b FILE]	True if FILE exists and is a block-special file.
[-c FILE]	True if FILE exists and is a character-special file.
[-d FILE]	True if FILE exists and is a directory.
[-e FILE]	True if FILE exists.
[-f FILE]	True if FILE exists and is a regular file.
[-g FILE]	True if FILE exists and its SGID bit is set.
[-h FILE]	True if FILE exists and is a symbolic link.
[-k FILE]	True if FILE exists and its sticky bit is set.
[-p FILE]	True if FILE exists and is a named pipe (FIFO).
[-r FILE]	True if FILE exists and is readable.
[-s FILE]	True if FILE exists and has a size greater than zero.
[-t FD]	True if file descriptor FD is open and refers to a terminal.

Table 8.2: Additional Primary expressions

Primary	Meaning
[-u FILE]	True if FILE exists and its SUID (set user ID) bit is set.
[-w FILE]	True if FILE exists and is writable.
[-x FILE]	True if FILE exists and is executable.
[-O FILE]	True if FILE exists and is owned by the effective user ID.
[-G FILE]	True if FILE exists and is owned by the effective group ID.
[-L FILE]	True if FILE exists and is a symbolic link.
[-N FILE]	True if FILE exists and has been modified since it was last read.
[-S FILE]	True if FILE exists and is a socket.

Table 8.3: Primary expressions

Primary	Meaning
[FILE1 -nt FILE2]	Newer than or if FILE1 exists and FILE2 does not.
[FILE1 -ot FILE2]	Older than FILE2, or if FILE2 exists and FILE1 does not.
[FILE1 -ef FILE2]	True if FILE1 and FILE2 refer to the same device and inode numbers.
[-o OPTIONNAME]	True if shell option “OPTIONNAME” is enabled.
[-z STRING]	True of the length if “STRING” is zero.
[-n STRING]	True if the length of “STRING” is non-zero.
[STRING1 == STRING2]	True if the strings are equal.
[STRING1 != STRING2]	True if the strings are not equal.
[STRING1 < STRING2]	True if “STRING1” sorts before “STRING2”
[STRING1 > STRING2]	True if “STRING1” sorts after “STRING2”
[ARG1 OP ARG2]	“OP” is one of -eq, -ne, -lt, -le, -gt or -ge.

Table 8.4: Boolean Operators

Boolean	Value
&&	Logical AND
	Logical OR
-a	Primary Expression AND
-o	Primary Expression OR

8.3.4.1 IF, ELSE, ELIF

If statements in BASH have support for nested IFs, IF ELSE constructs, and even [CASE](#) statements. Here is an example of a nested IF statement using Else IFs from the TLDP project. We will not cover the scope of CASE statements in this book - see the previous link for a good tutorial. Note at this level of complexity it might be better to try to engineer a CASE statement to re-architect what you are trying to do in to smaller steps to reduce complexity. Complexity is the enemy of the programmer.

```
#!/bin/bash
```

```
if [ -a ~/Documents ]
then
    echo "The documents directory exists"
else
    echo "Directory doesn't exist"
fi
```

```
#!/bin/bash
```

```
if [ -a ~/Documents ]
then
    echo "The documents directory exists"
else
    # This will fail silently
    echo "Directory doesn't exist" &> ~/Documents/script.log
fi
```

```
#!/bin/bash
```

```
if [ $# -gt 5 ]
then
    echo "You need to type only 5 positional parameters - you entered more!"
else
    echo "Good job you typed in 5 parameters"
fi
```

```
#!/bin/bash
```

```
if [ -a $1 ] && [ -x $1 ]
then
    echo "File exists and is executable"
else
    echo "File does not exist and/or is not executable"
fi
```

Here is a complex example of nested IF, ELIFs, and ELSE statements.

```
#!/bin/bash
```

```
# This script lets you present different menus to Tux. He will only be happy
# when given a fish. We've also added a dolphin and (presumably) a camel.
```

```

if [ "$menu" == "fish" ]; then
  if [ "$animal" == "penguin" ]; then
    echo "HMMMMMM fish... Tux happy!"
  elif [ "$animal" == "dolphin" ]; then
    echo "Pweetpeettreetppeterdepweet!"
  else
    echo "*prrrrrrrrt*"
  fi
else
  if [ "$animal" == "penguin" ]; then
    echo "Tux don't like that. Tux wants fish!"
    exit 1
  elif [ "$animal" == "dolphin" ]; then
    echo "Pweepwishpeeterdepweet!"
    exit 2
  else
    echo "Will you read this sign?!"
    exit 3
  fi
fi

```

8.3.4.2 CASE Statement

Sometimes called a switch statement in C or Java, a **case statement** is really just a way to simplify complicated nested IF statements ². []() statements. Here is an example of a nested IF statement using Else IFs from the TLDP project. We will not cover the scope of CASE statements in this book but I wanted to make you aware of them.

```

case "$1" in
  start)
    start
    ;;

  stop)
    stop
    ;;

  status)
    status anacron
    ;;
  restart)
    stop
    start
    ;;
  condrestart)
    if test "x`pidof anacron`" != x; then
      stop
      start
    fi
    ;;

  *)
    echo $"Usage: $0 {start|stop|restart|condrestart|status}"
    exit 1
esac

```

²http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_03.html

8.3.4.3 While Loops

This while loop will read each line of a text file allow you to operate on each value read. The file, roster.txt is located in the files > Chapter-08 > lab.

```
#!/bin/bash
# Assume the names.txt file exists from previous examples

while read LINE
do echo $LINE; mkdir -v /tmp/$LINE
done < names.txt
```

8.3.4.4 FOR Loops

A FOR loop is used to loop incrementally through a list until the end is met. In Bash the only data structure that you will use loop through are arrays and lists. Lists are not a datatype like in C and Java but simply a space delimited list of items. The syntax of a FOR loop is:

```
for arg in [LIST];
do
# code here
done
```

If the do portion of the for loop is on the same line as the for loop a single ; is needed between them.

```
#!/bin/bash

# Assuming the existence of the names.txt file
# from the previous Bash Array example

for NAME in ${NAMESARRAY[@]};
do
echo "The names in the file: $NAME"
done

#!/bin/bash
# Or you can list elements out manually

for planet in Mercury Venus Earth Mars Jupiter Saturn Uranus Neptune Pluto
do
echo $planet # Each planet on a separate line.
done

#!/bin/bash
# This is a traditional C based for loop

declare -a planetARRAY
planetARRAY=( mars earth mercury )
LENGTH=${#planetARRAY[@]}
echo "ARRAY LENGTH IS $LENGTH"

for (( i=0; i<${LENGTH}; i++));
do
echo ${planetARRAY[$i]}
sleep 1
done

# This is a valid 1 line for loop that prints out a single dot indicator
echo -e "\nFinished launching and sleeping 25 seconds"
for i in {0..25}; do echo -ne '.'; sleep 1;done
echo "\n"
```


8.4 Scheduling Shell Scripts With Cron

Now that we have sufficiently complex shell scripts the idea of automating their execution come into play. The concept of the **cron** command and the **crontab** files first came into use in Unix System V release ~1983. Each user can set their own scheduled tasks by editing the **crontab** file by typing **crontab -e**. The contents of the crontab file are initially blank. The language of the crontab is that of 5 columns and then a command to be executed. Multiple commands can be executed using **;** or **&&** And multiple different times can be listed in the crontab. The five (and sometimes a sixth fields) are as follows:

Time Unit	Values
minute	0 - 59
hour	0 - 23
day of month	1 - 31
month	1 - 12
day of week	0 - 6 (0 is Sunday)

The cron service scans for your combination of jobs and if a pattern matches the current time then that job executes. For example:

```
* 20 * * * /home/jeremy/backup.sh
```

In this case the script above would execute every night of the week at the 20th hour ³. The values of cron can be combined to get specific times and dates. There is also syntax for executing on every 15 minutes.

```
*/15 * * * * ~/Documents/script.sh
```

```
0,15,30,45 * * * * ~/Documents/script.sh
```

Something unique like this will start at 3 minutes after and then execute every 10 minutes (13,23,33,43,53 minutes)

```
3/10 * * * * ~/Documents/script.sh
```

```
@monthly ~/Documents/backup-data-to-off-site.sh
```

Any command that is executed obeys general shell meta-characters and variables as if you were typing that command directly on the command line. This also allows for redirection of standard out and standard error as well.

Some cron implementations support the following non-standard macros⁴:

Entry	Description	Equivalent to
@yearly	Run once a year at midnight of January 1st	0 0 1 1 *
@monthly	Run once a month, midnight 1st day of the month	0 0 1 * *
@weekly	Run once a week at midnight on Sunday morning	0 0 * * 0
@daily	Run once a day at midnight	0 0 * * *
@hourly	Run once an hour at the beginning of the hour	0 * * * *
@reboot	Run at startup	@reboot

8.4.1 Where to find more

O'Reilly media has many old but good books about Bash, vi, and shell scripting.

- Bash <http://shop.oreilly.com/product/9780596009656.do>
- Bash Cookbook <http://shop.oreilly.com/product/0636920058304.do>
- vi and vim <http://shop.oreilly.com/product/9780596529833.do>

³<https://en.wikipedia.org/wiki/Cron#Examples>

⁴https://en.wikipedia.org/wiki/Cron#Nonstandard_predefined_scheduling_definitions

8.5 Text Processing with Awk and Sed

8.5.1 AWK

AWK is a programming language designed for text processing and typically used as a data extraction and reporting tool. It is a standard feature of most Unix-like operating systems⁵. This original program was released in 1977, and was a powerful and focused language. We take things like Perl, Python, and even databases such as MySQL and SQLite. The `awk` language was designed to do all these things because at the time text based files was the universal datatype. The initial program was developed by [Alfred Aho](#), [Peter Weinberger](#), and [Brian Kernighan](#) at Bell Labs.

An AWK program consists of a sequence of optional directives, pattern-action statements, and optional function definitions.

```
awk pattern { action statements }
awk [options] <'program'> [file1] [file2][...]
```

Lets compare what `awk` can do:

- How would you find which ip caused the most HTTP 404 errors? Take these two files in `files/Chapter-08/logs` `u_ex150721.log` `u_ex151002.log`.
- How would we capture the top 5 offending IPs? What column number is `sc-status`?
- How could we look for everything that isn't a 404?
- How would you check for WordPress filesystem hacks by searching for `'/'` or `'.'` or `/etc` or names like `passwd.htaccess` `my.cnf`?
 - `awk '$11~/\.\.\.\/' u_ex150721.log`
- What is the difference between `cat hosts.deny` and `awk '{print;}' hosts.deny`
- How would you print out all lines of a file that contain a `#` as the first character?
 - `awk '$0~/^#/' hosts.deny`
- How would you print out all lines of a file that do not contain a `#` as the first character?
 - `awk '$0!~/^#/' hosts.deny`

The `awk` program works very well, but as the standard text based logs migrate to the binary format of `journalctl` what happens? The `journalctl` command has a `-no-pager` option to effectively print out all of the journal. You can use `awk` in conjunction. How would you scan the journal looking for all log entries related to the `sshd` daemon? How would you do it in `grep` and `cut`? How would you do it in `awk`?

```
sudo journalctl --no-pager | awk '$5 ~ /^sshd/'
```

Lets take a look at this statement:⁶

- `$5` tells AWK to look at the fifth “column”.
- `~` tells AWK to do a RegularExpression match `/. . . /` is a Regular expression.
- Within the RE is the string `Linux` and the special character `^`.
- `^` causes the RE to match from the start (as opposed to matching anywhere in the line).
- Seen together: AWK will match a regular expression with “Linux” at the start of the first column.

You can find more information at this IBM tutorial: [AWK by example](#). The GNU version of AWK is called `gawk` and it performs the same commands and options but is licensed under GPLv3+ as AWK was part of the proprietary UNIX tools and license. Linux systems symlink `awk` command to `gawk`.

8.5.2 sed

`sed` is a stream editor. A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline). While in some ways similar to an editor which permits scripted edits (such as `ed`), `sed` works by making only one pass over the input(s), and is consequently more efficient. But it is the ability of `sed` to filter text in a pipeline which particularly distinguishes it from other types of editors.

```
sed script inputfile
```

The `sed` command uses regular expressions between the `/. . /`⁷

⁵<https://en.wikipedia.org/wiki/AWK>

⁶<https://unix.stackexchange.com/questions/72734/awk-print-line-only-if-the-first-field-start-with-string-as-linux1>

⁷<https://www.ibm.com/developerworks/library/l-sed1/index.html>

- `^` Matches the beginning of the line
- `$` Matches the end of the line
- `.` Matches any single character
- `*` Will match zero or more occurrences of the previous character
- Matches all the characters inside the `[]`

The command to show all lines that do not start with a `#` below, how would you make the command print all the lines that do start with a `#`?

```
sed -e '/^#/d' hosts.deny | head
```

Looking at this command, using the GNU version of `sed` they introduced the `-i` flag that will make your change and re-write it to the same file. What is happening in the script below?

```
sudo sed -i "s/bantime = 600/bantime = 10000/g" /etc/fail2ban/jail.conf
```

For more information see this IBM tutorial: [sed by example](#)

8.6 Chapter Conclusions and Review

Through this chapter we learned about how to extend and improve our ability to write shell scripts. We learned about system variables and their scope. We also learned about positional parameters and how to pass values into a shell script. We learned about basic Bash control structures IF and FOR, and finally concluded learning about automating shell script launches using the cron service via cron tab.

8.6.1 Review Questions

- 1) True or False The Bash shell scripting language has traditional language constructs like C or Java?
- 2) What meta-character do you use to access the content of a shell variable?
 - a. `$`
 - b. `#`
 - c. `!`
 - d. No character - trick questions
- 3) When assigning the standard output of a command to a variable which of these are valid methods?
 - a. `DT=`date``
 - b. `DT=$(date)`
 - c. `DT="date"`
 - d. Not possible, trick questions
- 4) What is the name of the program used for manipulating streams of text inside of a file (used for find and replace)?
 - a. `awk`
 - b. `grep`
 - c. `pgrep`
 - d. `sed`
- 5) What would be the correct syntax to find and replace the value 127.0.0.1 with 0.0.0.0 in the file `50-server.cnf`?
 - a. `sudo sed "s/0.0.0.0/127.0.0.1/g" 50-server.cnf`
 - b. `sudo sed -s "s/0.0.0.0/127.0.0.1/g" 50-server.cnf`
 - c. `sudo sed -i "s/0.0.0.0/127.0.0.1/g" 50-server.cnf`
 - d. `sudo sed -r "s/0.0.0.0/127.0.0.1/g" 50-server.cnf`
- 6) Which of these are valid commands in the first line of a shell script? (Choose any - assume any paths are valid paths to executables)
 - a. `#!/bin/bash`
 - b. `!#/bin/bash`
 - c. `#!/usr/local/bin/bash`

- d. `#!/bin/bash`
 - e. `#!/bin/sh`
- 7) If you stored the output of the command `hostname` into a variable named `ROSTER`, what would be the command to print the content to the screen?
- a. `echo $ROSTER`
 - b. `echo $roster`
 - c. `echo ROSTER`
 - d. `echo roster`
- 8) What is the proper syntax to end an IF statement – what is the very last line?
- a. `}`
 - b. `done`
 - c. `fi`
 - d. `exit`
- 9) What is the correct **primaries** syntax for an IF statement, to check if a file exists?
- a. `if [-e filename]`
 - b. `if [-e filename]`
 - c. `if (-e filename)`
 - d. `if [true filename]`
- 10) Which of these is a valid command to redirect the standard out of a `cat` command to a shell variable?
- a. `NAMES = (cat roster.txt)`
 - b. `NAMES << (cat roster.txt)`
 - c. `NAMES=($(cat roster.txt))`
 - d. `NAMES < (cat roster.txt)`
- 11) Which below is a valid command to find the `LENGTH` of an array?
- a. `${#SEARCHARRAY[@]}`
 - b. `${SEARCHARRAY[@]}`
 - c. `${SEARACHARRAY[#]}`
 - d. `${@SEARCHARRAY[#]}`
- 12) Based on this shell script and positional parameters, what would the command be to print out the first positional parameter after the script name? `./delete-directory.sh ~/Documents/text-book Jeremy`
- a. `echo $0`
 - b. `echo $1`
 - c. `echo args[1]`
 - d. `echo ${1}`
- 13) Based on this shell script and positional parameters, what would the command be to print out the entire content of the positional parameter array? `./delete-directory.sh ~/Documents/text-book Jeremy`
- a. `echo $#`
 - b. `echo @!`
 - c. `echo $0`
 - d. `echo $@`
- 14) Based on this shell script and positional parameters, what would the command be to print out the `LENGTH` of the positional parameter array? `./delete-directory.sh ~/Documents/text-book Jeremy`
- a. `echo $#`
 - b. `echo @!`
 - c. `echo $0`
 - d. `echo $@`
- 15) In a Bash IF statement, what is the name for the pre-made test conditions?
- a. Primaries

- b. Secondary expressions
 - c. Expression
 - d. Primary expressions
- 16) All values in a Bash IF statement are of what data type by default?
- a. INT
 - b. STRING
 - c. NULL
 - d. CHAR
- 17) Which of these answers will execute a shell script named `~/backup.sh` at 2 am every night of the week?
- a. `* * * * * ~/backup.sh`
 - b. `2 * * * * ~/backup.sh`
 - c. `* 2 * * * ~/backup.sh`
 - d. `* * * 2 * ~/backup.sh`
- 18) Which of these answers will execute a shell script named `~/clean-directory.sh` every 15 minutes?
- a. `3/15 * * * * ~/clean-directory.sh`
 - b. `*/15 * * * * ~/clean-directory.sh`
 - c. `* 3/15 * * * ~/clean-directory.sh`
 - d. `* */15 * * * ~/clean-directory.sh`
- 19) Which of the crontab builtins would you use to execute a cron job 1 time a year on midnight of January 1st?
The name of the script is `/home/controller/homework-is-finished.sh`
- a. `* * * * 1 /home/controller/homework-is-finished.sh`
 - b. `1 * * * * /home/controller/homework-is-finished.sh`
 - c. `1 1 1 1 1 /home/controller/homework-is-finished.sh`
 - d. `@yearly /home/controller/homework-is-finished.sh`
- 20) What is the name of the control structure that allows you to incrementally through the contents of an array?
- a. IF
 - b. CASE
 - c. UNTIL
 - d. FOR

8.6.2 Podcast Questions

Watch or listen to this Podcast: OwnCloud <https://twit.tv/shows/floss-weekly/episodes/274> “OwnCloud Podcast Floss Weekly)

- ~2:50 - Who is the creator of OwnCloud?
- ~3:23 - How is OwnCloud’s purpose described?
- ~4:03 - Is OwnCloud opensource and can you install it on your own hardware?
- ~5:20 - What other opensource project is the guest involved in?
- ~5:40 -6:55 - According to the guest speaker, what is wrong with Google Drive and DropBox?
- ~7:56 - According to the Host – why do people not run their own instances of OwnCloud?
- ~8:28 - Using a hosted OwnCloud Provider what is the advantage with moving your data that other companies don’t give you?
- ~12:28 - Can OwnCloud connect to a University authentication system?
- ~12:44 - What is the largest install base for a University using OwnCloud?
- ~15:07 - What language is the server written in?
- ~15:27 - What language and library are the Linux/MacOS/Windows sync clients built in?
- ~20:55 - What does the c-sync tool do for OwnCloud?
- ~23:58 - How many contributors to code does OwnCloud have?
- ~25:21 - What is the most important thing to have in order to build a thriving opensource community, according to the guest?
- ~26:45 - Is there commercial support available for OwnCloud?

- ~27:44 - What is the OpenSource License the OwnCloud project using?
- ~28:28 - What rights to the GNU AGPL enforce/give you?
- ~31:00 - What guarantee does a code contributor have about anything they submit to the OwnCloud project?
- ~34:12 - How safe can a web-hosted OwnCloud system be against hackers?
- ~37:30 - What does the FreedomBox project do (<https://www.freedombox.org/>) and how do they compare OwnCloud?
- ~41:30 - What is the main new Google-like feature discussed/added in OwnCloud 6?

8.6.3 Lab

8.6.3.1 Lab Objectives

This lab will allow you to create shell scripts. Use positional parameters, control structures, and write cron jobs.

8.6.3.2 Lab Outcomes

At the completion of this lab you will further your knowledge of shell scripting and enhance your abilities using Bash shell scripts.

- 1) Create a shell script that will take 3 positional parameters, append each parameter to a file named roster.txt (each of the parameters will be a name).
- 2) Create a shell script to cat the content of the roster.txt file into an array named: ROSTERARRAY and echo the 2nd element of the array.
- 3) Create a shell script that redirects the content of roster.txt into an array, uses a forloop to cycle through the array's contents, and then uses the command to make a directory for each name listed in the array in your home directory, echo a message telling the user the path of the directory just created. Final command of the script is to list the content of the home directory to show the success of the script.
- 4) Modify the shell script from the previous question to include an if statement that checks for the existence of a directory. If the directory exists, echo a message: "Directory \$NAME exists". Then add an else clause that if the directory does not exist, create it and echo a message that the new directory and its path have been created.
- 5) Write a WHILE loop that will read the content of the file names.txt, (located in the files > Chapter-08 > lab folder) and create a directory based on the value on the line in the /tmp directory (one per users). Include an if statement to detect if the directory already exists, if it does exist, write the duplicate name out to a text file named: duplicates.txt located in the /tmp directory.
- 6) Write the syntax to make a cronjob execute 5 minutes past every hour everyday to execute the shell script you previously made to store the content of `ls -l ~` into an array named DIRARR.
- 7) Locate the file `install-java8.sh` located in the files > Chapter-08 > lab directory. Modify the script to include IF statements to check for the existence of the path /datapool1 and to print an error message if the path does not exist.
- 8) Modify `install-java8.sh` again—this time take a positional parameter and put that in place of the directory name /datapool1 (this will allow you to customize the install location of the shell script).
- 9) Create a directory in ~ named `topsecret`. In that directory create a file named `xfile.txt`. Write a shell script to check if that file has executable permission by passing the filename as a positional parameter. If TRUE print a message. If FALSE print an error message that the positional parameter name of the file is not executable.
- 10) Write a shell script to check in the ~/topsecret directory to see if a given file name exists (passed in by positional parameters). If TRUE print a message else print an error message with the given file name being passed.
- 11) Write a shell script to check if a given PATH, via a positional parameter, \$1, is a file or a directory. If TRUE print a message, else print an error message using the given file name.
- 12) Write a shell script that takes 4 positional parameters. In the shell script print out \$0, \$# , and \$@ with an explanation of what these variables contain.

- 13) Write a `cron` job that executes the command, `sudo mysqldump --all-databases` at 11:59 pm on Sunday every week of the month.
- 14) Using `awk` and other tools, how would you find which ip caused the most HTTP 404 errors? Take a screenshot of the command and the output. Use these two files in `files/Chapter-08/logs`: `u_ex150721.log`, `u_ex151002.log`.
- 15) Using `awk` and other tools, how would you capture the top 5 offending IPs? Take a screenshot of the command and the output. Use these two files in `files/Chapter-08/logs`: `u_ex150721.log`, `u_ex151002.log`.
- 16) Using `sed`, type the command to find the line `bind-address` located in the mariadb database server config file (you might need to install mariadb-server). The file locations are: Fedora `/etc/my.cnf.d/mariadb-server.cnf` and Ubuntu `/etc/mysql/mariadb.conf.d/50-server.cnf`. Comment out the value, change the IP value to 0.0.0.0, and write the change back to the original file. Take a screenshot of the output.

8.6.3.3 Footnotes

Chapter 9

System Administration

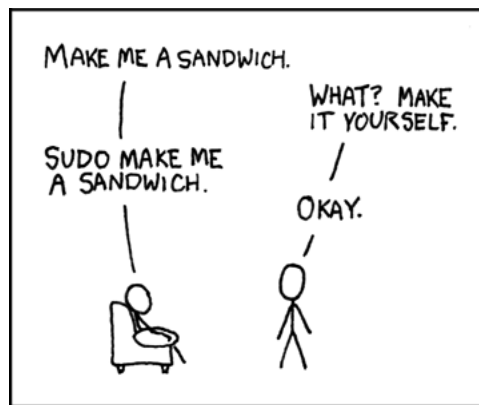


Figure 9.1: *This generation's 'Who's on First?'*

9.1 Objectives

- Understand the concept of the `sudo` command and the root user security implications
- Understand the basic admin tools and the Linux concept of logging and the systemd concept of logging with `journald`
- Understand how to use basic system tools for system monitoring and reporting
- Understand using standard tools for user administration
- Understand the 3Ps method of troubleshooting

9.2 Outcomes

At the completion of this chapter you will have the ability to administer a Linux system. You will have an understanding of Linux system logs, their standard locations, and their use. You will have a knowledge of system monitoring tools and how to understand their output. You will be able to administer user accounts on a Linux system. Finally you will be able to perform trouble shooting procedures on a Linux system.

9.3 Sudo and the Root User Paradigm

On every Unix system dating back to Thompson's Unix, there has always been a single *superuser* account on every system. This account is usually called the **root user** or **root**. The **root user** must be used with the utmost care, as that account has all the system privilege and authority to carry out any operation, even the `rm -rf /*` command.

Root is good for getting things done or overriding users, but is *dangerous*. You should log into that account only sparingly. Every single admin worth their salt will tell you not to use **root** in almost any case ¹.

This concept is *vital* enough that on the Ubuntu distribution there is no **root** account available. On the BSD distros, Debian, and the Red Hat/Fedora family - there is still a **root** account, partially because of tradition and partially because of the way system administration always worked. Remember that Unix was developed in the environment of multiple users accessing a large central Unix server. So you always had to have a **root** account to override any activities of the users and to enforce system policies, such as disk quotas, changing network configurations, or even system wide profiles. As a reminder when you are signed in as, or acting as “root”, the shell prompt displays **#** as the last character in bash and as seen in the image below. You can use the **whoami** command to find out what user account you are logged in as well.

```
[controller@localhost ~]$ whoami
controller
[controller@localhost ~]$ su root
Password:
[root@localhost controller]# whoami
root
[root@localhost controller]#
```

Figure 9.2: Root User has the # sign as its shell

In order to change the user you are logged in as, without logging out, you use the **su** command. Known as *superuser* or *switch user*. By typing **su root** on the command line in Fedora Linux, you will be presented with a password prompt to enter the password you created during the install process for the root account. You still need to type **exit** to logout from the user you switched too. The **sudo** command on the other hand, only elevates you for a default period of 15 minutes and then returns you to your standard user account. It is good habit to type **sudo** in front of any command that needs elevated privileges.

9.3.1 sudo

As a great philosopher once said, “*With great power comes great responsibility.*” Seeing as **root** has unintentionally dangerous uses a temporary system was devised to blunt the power of the **root** account. The **sudo** command was created by researchers at SUNY/Buffalo in New York in 1980 to allow users to run specific commands as a different user, in this case as root while not remaining or needing to sign in as root.

From 1986 to 1991, development of **sudo** moved to CU-Boulder in Colorado and gained the **cu-sudo** prefix. In 1991, the code was relicensed under the GPLv2. In 1996 Todd C. Miller (one of the early maintainers) took the project under his wing moving a version of **sudo** to his own servers, to differentiate from **cu-sudo**. By 1999 the code base was moved to the ISC license ([Internet Systems Consortium](http://www.isc.org)), the same license the **bind-dns** server is under, it is the preferred license of the OpenBSD project and is GPL compatible free license. Todd C. Miller is paid by Dell to maintain **sudo** as part of his day job. The **sudo** project homepage is located at <http://www.sudo.ws>. [A brief history of sudo](#) ². The tool is often mispronounced “*su - doh*”. But actual pronunciation is “*su - doo*”. You can learn more about **sudo** implementation and security in a video by Michael Lucas entitled, [Sudo: You’re doing it wrong](#).

9.3.1.1 Ubuntu

Ubuntu is a bit different from the other Linux and Unix distros in regards to **sudo**. They firmly believe not to have a root account as a point of differentiation. They rely on **sudo** hence the cartoon above. The first user you create (like in Windows and Mac) is automatically added to the **sudo** usergroup and has **sudo** privilege. Then any command you need *superuser* privileges you can simply elevate to that privilege by typing the word **sudo** in front on any command. Upon successful entry of your own password you will be elevated up to full system authority. Some refer to this as *god mode* but I think using that term is a bit presumptuous as you do have absolute power over the system but **sudo** doesn’t let you create the world in seven days.

One example is you can assign the permission value of 000 to a file. Who can access that file now? According to the permissions, not even the owner can access it. But the root user can, or a user issuing a **sudo** command. You can find

¹<http://www.tldp.org/LDP/lame/LAME/linux-admin-made-easy/root-account.html>

²<http://www.sudo.ws/history.html>

which users on a system have sudo permissions by displaying the `/etc/sudoers` file: `cat /etc/sudoers` (you need sudo privilege). Here is a sample screen shot where you define which users can be in the sudo group. You may not want to give admin privilege to every user. The conf file, under the `user` section, allows you to specify root privileges per command.

```
Defaults    env_reset
Defaults    mail_badpass
Defaults    secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:$"
```

Figure 9.3: *Ubuntu /etc/sudoers*

```
# User privilege specification
root    ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges
%admin  ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo  ALL=(ALL:ALL) ALL
```

Figure 9.4: *Ubuntu /etc/sudoers*

Let's look at the contents in more detail. First to edit the `/etc/sudoers` file you do not directly edit the file, but through a special tool called `visudo`. The `visudo` command edits the sudoers file in a safe fashion. The command `visudo` locks the sudoers file against multiple simultaneous edits, provides basic sanity checks, and checks for parse errors. If the sudoers file is currently being edited you will receive a message to try again later ³. You can invoke `visudo` from anywhere on the system.

The first line is where you set the system path a user receives when they become a sudo user.

```
Defaults    secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
```

Example Usage: Using either Fedora or Ubuntu as your user account from the commandline type `echo $PATH` now type `sudo sh` and then `echo $PATH` notice what happens to the prompt? Are the paths different? Why? Type `exit` to exit back to the normal user.

The next line of interest is:

```
# User privilege specification
root    ALL=(ALL:ALL) ALL
```

This line allows you to add specific users and then list specific commands that they have *superuser* access too.

```
# Allow members of group sudo to execute any command
%sudo  ALL=(ALL:ALL) ALL
```

The next line allows you to add groupnames to receive sudo access. Any user account that is a member of this group, in Ubuntu's case *sudo* can gain *superuser* permissions

```
# Members of the admin group may gain root privileges
%admin  ALL=(ALL:ALL) ALL
```

This last entry is a catch-all command for backward compatibility. Ubuntu versions previous to 14.04 used the *admin* group instead of the *sudo* group.

9.3.1.2 sudoers values

What do the values `%admin ALL=(ALL:ALL) ALL` mean? This particular command gives every user in the *admin* group access to execute every command the system. It essentially turns the root account *superuser* privileges over

³<http://manpages.ubuntu.com/manpages/dapper/man8/visudo.8.html>

temporarily to the user account or group that has that privilege.

The first column is either a user account (no %) or preceded by a % sign meaning a user group. The second column (or the first ALL) is the hostname of the systems that can allow elevation to *superuser*. Now if this is your only system the value can be left at ALL. But if you are preparing enterprise-wide */etc/sudoers* configurations then you may want to specify *superuser* access only on particular systems. The third column (second ALL) is the user that you will turn into when you use the *sudo* command. By default it is **root** but you may want it to be another specific user. The fourth column after the : (third all) is the comma separated list of commands that user can execute. The fifth column (fourth ALL) is optional but it is an access control feature allowing only members of certain groups to sudo.

Group	Hosts	Target User	Commands
%sudo	server-1, db-server	bkupuser	/usr/sbin/postfix /usr/sbin/doveadm
%admin	ALL	dbadmin	ALL
%cia	ALL	ALL	ALL

After the useraccount you can add an additional parameter to remove the password requirement. This is dangerous because it means anyone who has local access to the system can now become a *superuser* account just by switching users. It is best to leave this task for remote automated users or narrow down the powers to a single specific job.

Example Usage: The two commands below give the user **bkupuser** the ability to become sudo without requiring a password and only the power to execute the copy command. The second command gives any user who is a member of the admin group the ability to sudo with out any password.

```
bkupuser ALL=(root) NOPASSWD: /usr/bin/mysqldump
%vagrant ALL=(ALL) NOPASSWD:ALL
%admin ALL=(ALL) NOPASSWD:ALL
```

9.3.1.3 Fedora and other Linux

All other Linux distributions have a **root** account user made at install time. Some minimal distributions or in FreeBSD case may only allow you to create a **root** user at install time and make additional users your job to create. In Fedora you can log into an GNOME session using the root account, there might be warnings from the operating system, as it is not expecting you to be logged in as **root**. The **root** user has its own home directory located at */root*. Even if you are going to use the **root** account it is still advised to log in as a regular user and then use the *sudo* or *su* commands to elevate and then exit those privileges.

Fedora and other Linux/Unix/Mac use different groups for *sudo* and *superuser* access. That group is called *wheel*. If you look at the */etc/sudoers* output below from Fedora system you see the groups and file content is slightly different.

```
## Allow root to run any commands anywhere
root    ALL=(ALL)        ALL

## Allows members of the 'sys' group to run networking, software,
## service management apps and more.
# %sys  ALL = NETWORKING, SOFTWARE, SERVICES, STORAGE, DELEGATING, PROCESSES, LOCATE, DRIVERS

## Allows people in group wheel to run all commands
%wheel  ALL=(ALL)        ALL

## Same thing without a password
# %wheel    ALL=(ALL)        NOPASSWD: ALL

## Allows members of the users group to mount and unmount the
## cdrom as root
# %users  ALL=/sbin/mount /mnt/cdrom, /sbin/umount /mnt/cdrom

## Allows members of the users group to shutdown this system
# %users  localhost=/sbin/shutdown -h now
```

Figure 9.5: *Fedora /etc/sudoers*

9.3.1.4 sudo usage examples and conclusion

Example Usage: After installing the apache webserver (httpd) on Fedora - the html files are served out of the default directory `/var/www/html`. Now if you `cd` to that `/var/www/` what do you notice about group and other ownership of `html`? How would you write a newfile `support.html` in that directory?

Example Usage: To install a service: `sudo dnf install httpd`, then you need to start the service (on Ubuntu they autostart for you, Fedora family they don't autostart), `sudo systemctl start httpd`.

That is sudo in a nutshell, be careful and happy sudo-ing. To learn more about the heavy details of sudo you can watch [this presentation http://blather.michaelwlucas.com/archives/2266](http://blather.michaelwlucas.com/archives/2266) from Michael Lewis.

9.4 Logging and Monitoring



One of the most central functions of an operating system is logging. Without logging facilities it would be difficult to keep track of what the system is doing. The technical term for this is **introspection**. In the course of your Linux career you will find the logging system to be of immense help. Not only can it be used to debug problems and find errors or security issues, but also to monitor and measure that changes made to the operating system are working to prevent issues. From here on out when there is an application problem in Linux - your first trouble shooting step should be to go to the logs.

9.4.1 `/var/log/*`

The default logging directory on all Linux systems is located in `/var/log`. This is the place where the kernel, the operating system, and any default installed services will place its logs. For 30+ years this was the convention and all common knowledge. But with the recent adoption of `systemd` on all major Linux platforms, the logging facility that was once simple text, has now been moved into the `journald` and into a binary format. Note with the `systemd`, the logging convention has been changed to a binary format and placed under the `journalctl` command which we will cover in chapter 11.

When you install additional packages, those packages too will add a directory for its own logs. Note in the picture below there is a log called `httpd` that is created when you install the `https` (apache2 webserver package) to track the `webserver error.log` and `access.log` files. You will notice in these screenshots that there is a log entry for `VBoxGuestAdditions`—telling you that you are using `VirtualBox`.

⁴“PONDEROSA PINE LOGS STACKED AT PINE INDUSTRY MILL - NARA - 542596” by Daniels, Gene, photographer, Photographer (NARA record: 8463941) - U.S. National Archives and Records Administration. Licensed under Public Domain via Wikimedia Commons.

```

[root@localhost log]# ls
anaconda      dnf.librepo.log-20150930  dnf.rpm.log-20150930  hawkey.log-20151019  sssd
atop          dnf.librepo.log-20151019  dnf.rpm.log-20151019  hawkey.log-20151028  tallylog
audit         dnf.librepo.log-20151028  dnf.rpm.log-20151028  httpd                 vboxadd-install.log
boot.log      dnf.log                   firewallld             journal               vboxadd-install-x11.log
btmptmp       dnf.log-20150921          gdm                   lastlog               VBoxGuestAdditions.log
btmptmp-20151001  dnf.log-20150930        glusterfs             libvirt              wpa_supplicant.log
chrony        dnf.log-20151019         grubby                README               wtmp
cluster       dnf.log-20151028         hawkey.log            README               Xorg.0.log
cups          dnf.rpm.log              hawkey.log-20150921  samba                 Xorg.0.log.old
dnf.librepo.log  dnf.rpm.log-20150921    hawkey.log-20150930  samba                 speech-dispatcher
[root@localhost log]#

```

Figure 9.6: *Fedora contents of /var/log/**

```

controller@controller-VirtualBox:/var/log$ ls
alternatives.log      boot.log             fontconfig.log      ntpstats            unattended-upgrades
alternatives.log.1    bootstrap.log        fsck                 samba               upstart
alternatives.log.2.gz  btmptmp             gpu-manager.log     sddm.log           vboxadd-install.log
apache2               btmptmp.1           hp                   speech-dispatcher  vboxadd-install-x11.log
apport.log            ConsoleKit           installer           syslog              VBoxGuestAdditions.log
apport.log.1         cups                kern.log             syslog.1           wtmp
apt                   dist-upgrade        kern.log.1           syslog.2.gz        wtmp.1
auth.log              dmesg               kern.log.2.gz       syslog.3.gz        wvdialconf.log
auth.log.1            dpkg.log            kern.log.3.gz       syslog.4.gz        Xorg.0.log
auth.log.2.gz         dpkg.log.1          kern.log.4.gz       syslog.5.gz        Xorg.0.log.old
auth.log.3.gz         dpkg.log.2.gz       lastlog             syslog.6.gz        Xorg.1.log
auth.log.4.gz         faillog             lightdm              syslog.7.gz        Xorg.1.log.old

```

Figure 9.7: *Ubuntu contents of /var/log**

9.4.2 syslog

The operating system needs a convention on how all the logs are transferred and stored. That method was called syslog. Until 1980 there were various logging methods and schemes. The one that caught on was called syslog and was actually part of an email program, Sendmail, initially. Syslog permits the consolidation of logging data from different types of systems in a central repository. Syslog logs can also be transmitted remotely and aggregated on a central system. Originally the protocol used UDP to reduce network traffic, but now mandates the protocol to use TCP and even TLS. Syslog listens on port 514 and has no authentication mechanism, deferring to the user to allow or block access via the firewall or other network access control. Fedora removed syslog as standard back in Fedora 20 and moved to `journalctl`. The system logs that had been stored in: ⁵

⁵<https://fedoraproject.org/wiki/Changes/NoDefaultSyslog>

9.4.2.1 Error Levels

Syslog has adopted default levels to describe the severity of a log that is recorded. Standard syslog error levels ⁶.

Value	Severity	Keyword	Description	Examples
0	Emergency	emerg	System is unusable	This level should not be used by applications.
1	Alert	alert	Should be corrected immediately	Loss of the primary ISP connection
2	Critical	crit	Critical conditions	A failure in the system's primary application.
3	Error	err	Error conditions	An application has exceeded its file storage limit and attempts to write are failing.
4	Warning	warning	May indicate an error will occur if not taken.	A non-root file system has 2GB remaining.
5	Notice	notice	Events that are unusual, but not errors	Conditions.
6	Informational	info	Normal operational messages that require no action.	An application has started, paused or ended successfully.
7	Debug	debug	Information useful to developers for the application	debugging.

9.4.3 rsyslog

By the year 2004 the clear need for a syslog compatible but feature rich replacement was needed. Rsyslog was developed by [Rainer Gerhards](#) and in his words, “**Rsyslog is a GPL-ed, enhanced syslogd. Among others, it offers support for reliable syslog over TCP, writing to MySQL databases and fully configurable output formats (including great timestamps).**” It was an improvement on syslog. It made syslog extensible and eventually replaced syslog by default. Most Linux distributions dropped the original syslog application and replaced it with rsyslog by 2010 ⁷.

9.4.4 journald and systemd

Not to be outdone - `systemd` has replaced syslog with `journald`. And this has happened in every system that has adopted `systemd` - Debian 8, Fedora 22, Ubuntu 15.04/15.10, CentOS 7. You can read the initial `journald` announcement and [justification paper here](#) ⁸.

In Lennart Poettering's own words, “*If you are wondering what the journal is, here's an explanation in a few words to get you up to speed: the journal is a component of systemd, that captures Syslog messages, Kernel log messages, initial RAM disk and early boot messages as well as messages written to STDOUT/STDERR of all services, indexes them and makes this available to the user. It can be used in parallel, or in place of a traditional syslog daemon, such as rsyslog or syslog-ng.*” ⁹

“*One of the impetuses behind the systemd journal is to centralize the management of logs regardless of where the messages are originating. Since much of the boot process and service management is handled by the systemd process, it makes sense to standardize the way that logs are collected and accessed. The journald daemon collects data from all available sources and stores them in a binary format for easy and dynamic manipulation.*” ¹⁰

If you are using a version of RHEL 6, CentOS 6, Ubuntu 14.04, or Debian 7 and prior you will not find the `journald` or `systemd` commands and will find the traditional syslog service. Syslog can be installed along side of `journald` and

⁶<https://en.wikipedia.org/wiki/Syslog>

⁷<http://www.rsyslog.com/doc/history.html>

⁸<https://docs.google.com/document/pub?id=1IC9yOXj7j6cdLLxWEBAGRL6w197tFxxjLUEHIX3MSTs&pli=1>

⁹<http://0pointer.de/blog/projects/journalctl.html>

¹⁰<https://www.digitalocean.com/community/tutorials/how-to-use-journalctl-to-view-and-manipulate-systemd-logs>

run in the traditional sense. Some argue that this is a violation of the Unix principle of small services doing one thing (systemd is not small and does many things). Some even claim that the journald logging service is no different than the Windows Event Logger and the way in which Windows does logs. The traditional ways of using syslog had been modified by journald.

- `cat /var/log/messages` will now become `journalctl`
- `tail -f /var/log/messages` will now become `journalctl -f`
- `grep sshd /var/log/messages` will now become `journalctl -u sshd`

To use the journal daemon (journald) all its elements are accessed through the `journalctl` command. All previously sparse logs are now contained in a single binary append only log format. The advantage of that is that the output can be programmatically parsed (actually queried like a database) the downside is that some people see an “all your eggs in one basket” problem with a single central binary file.

Example Usage: These examples have been taken from the [systemd website](#): ¹¹

Without arguments, all collected logs are shown unfiltered:

```
journalctl
```

With one match specified, all entries with a field matching the expression are shown:

```
journalctl _SYSTEMD_UNIT=avahi-daemon.service
```

If two different fields are matched, only entries matching both expressions at the same time are shown:

```
journalctl _SYSTEMD_UNIT=avahi-daemon.service _PID=28097
```

If two matches refer to the same field, all entries matching either expression are shown:

```
journalctl _SYSTEMD_UNIT=avahi-daemon.service _SYSTEMD_UNIT=dbus.service
```

If the separator “+” is used, two expressions may be combined in a logical OR. The following will show all messages from the Avahi service process with the PID 28097 plus all messages from the D-Bus service (from any of its processes):

```
journalctl _SYSTEMD_UNIT=avahi-daemon.service _PID=28097 + _SYSTEMD_UNIT=dbus.service
```

Show all logs generated by the D-Bus executable:

```
journalctl /usr/bin/dbus-daemon
```

Show all kernel logs from previous boot:

```
journalctl -k -b -1
```

Show a live log display from a system service `apache.service`:

```
journalctl -f -u apache
```

This will show you only the logs of the current boot:

```
journalctl -b
```

List all messages of priority levels ERROR and worse, from the current boot:

```
journalctl -b -p err
```

Filtering based on time

```
journalctl --since=yesterday
```

Filter based on time range - note how difficult this would be with using `grep`, `sort`, and `awk` because everything is text. But since journald can be thought of a similar to a SQL database, then these types of queries are possible.

```
journalctl --since=2012-10-15 --until="2011-10-16 23:59:59"
```

See log entries created only by the SSH service

¹¹<http://www.freedesktop.org/software/systemd/man/journalctl.html>

```
journalctl -u ssh.service
```

9.4.5 Log rotation

The concept of log rotation existed under syslog and rsyslog when logs were simple text files, but no longer exists under journald. This was accomplished by the `/etc/logrotate.conf`. Applications may still write to a discrete syslog (or may not), but all logs are then copied by journald, which systemd made the central repository for logs.

When viewing an older syslog style text log you can use the `tail -f` command and it will auto-update if there is new content automatically. This command can be very helpful if you are watching a log for some particular output - can you find the journald equivalent? `journalctl --follow --since=-1day`

You can find the systemd journald log rotation and collection specifics [here](#): ¹².

```
/etc/systemd/journal.conf
```

Below are the default settings - even though they are commented out they are set there to give a template for a system admin to modify.

Entries can be service specific and kept in subdirectories `/etc/systemd/journal.conf.d/*.conf` any configurations in these directories take precedence over the main `journal.conf` file.

Storage Controls where to store journal data. One of “volatile”, “persistent”, “auto” and “none”. If “volatile”, journal log data will be stored only in memory, i.e. below the `/run/log/journal` hierarchy (which is created if needed). If “persistent”, data will be stored preferably on disk, i.e. below the `/var/log/journal` hierarchy (which is created if needed), with a fallback to `/run/log/journal` (which is created if needed), during early boot and if the disk is not writable. “auto” is similar to “persistent” but the directory `/var/log/journal` is not created if needed, so that its existence controls where log data goes. “none” turns off all storage, all log data received will be dropped. Forwarding to other targets, such as the console, the kernel log buffer, or a syslog socket will still work however. Defaults to “auto”.

SplitMode Controls whether to split up journal files per user. One of “uid”, “login” and “none”. If “uid”, all users will get each their own journal files regardless of whether they possess a login session or not, however system users will log into the system journal. If “login”, actually logged-in users will get each their own journal files, but users without login session and system users will log into the system journal. If “none”, journal files are not split up by user and all messages are instead stored in the single system journal. Note that splitting up journal files by user is only available for journals stored persistently. If journals are stored on volatile storage (see above), only a single journal file for all user IDs is kept. Defaults to “uid”.

MaxLevelStore, MaxLevelSyslog, MaxLevelKMsg, MaxLevelConsole, MaxLevelWall Controls the maximum log level of messages that are stored on disk, forwarded to syslog, kmsg, the console or wall (if that is enabled, see above). As argument, takes one of “emerg”, “alert”, “crit”, “err”, “warning”, “notice”, “info”, “debug”, or integer values in the range of 0..7 (corresponding to the same levels). Messages equal or below the log level specified are stored/forwarded, messages above are dropped. Defaults to “debug” for `MaxLevelStore=` and `MaxLevelSyslog=`, to ensure that the all messages are written to disk and forwarded to syslog. Defaults to “notice” for `MaxLevelKMsg=`, “info” for `MaxLevelConsole=`, and “emerg” for `MaxLevelWall=`.

SystemMaxFileSize and RuntimeMaxFileSize Control how large individual journal files may grow at maximum. This influences the granularity in which disk space is made available through rotation, i.e. deletion of historic data. Defaults to one eighth of the values configured with `SystemMaxUse=` and `RuntimeMaxUse=`, so that usually seven rotated journal files are kept as history.

Specify values in bytes or use K, M, G, T, P, E as units for the specified sizes (equal to 1024, 1024²,... bytes). Note that size limits are enforced synchronously when journal files are extended, and no explicit rotation step triggered by time is needed.

SystemMaxFiles and RuntimeMaxFiles Control how many individual journal files to keep at maximum. Note that only archived files are deleted to reduce the number of files until this limit is reached; active files will stay around. This means that in effect there might still be more journal files around in total than this limit after a vacuuming operation is complete. This setting defaults to 100.

¹²<http://www.freedesktop.org/software/systemd/man/journal.conf.html>

9.5 System Monitoring

The first step in system administration is monitoring. Just like viewing logs, also knowing what is currently going on resource wise can be very helpful. The first command we want to look at to help us understand what is occurring on our system is a command called `top`. This stands for *table of processes*. Top produces a list of running processes selected by user-specific criteria ¹³. The traditional Unix version was written by William LeFebvre and originally copyrighted in 1984. Since 1991 there has been a Linux based GPL top command which is part of the [procps-ng suite of tools](#) ¹⁴.

9.5.1 top

```
top - 13:57:27 up 58 min, 2 users, load average: 0.01, 0.09, 0.12
Tasks: 175 total, 2 running, 173 sleeping, 0 stopped, 0 zombie
%Cpu(s): 6.4 us, 0.7 sy, 0.0 ni, 92.9 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 2049520 total, 159148 free, 853616 used, 1036756 buff/cache
KiB Swap: 1888252 total, 1887768 free, 484 used. 996520 avail Mem
```

PID	USER	PR	NI	VRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1776	control+	20	0	1531748	202716	74984	S	3.3	9.9	2:21.08	gnome-shell
1595	root	20	0	286592	71384	31044	S	3.0	3.5	0:42.81	Xorg
2348	control+	20	0	613512	34720	26156	S	1.7	1.7	0:12.02	gnome-terminal-
3381	control+	20	0	1054136	225356	89436	S	1.0	11.0	0:30.24	firefox
3578	root	20	0	0	0	0	S	0.3	0.0	0:00.10	kworker/0:0
3672	control+	20	0	159820	4504	3804	R	0.3	0.2	0:00.03	top
1	root	20	0	129048	8976	4144	S	0.0	0.4	0:02.62	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:00.28	ksoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:0H
7	root	20	0	0	0	0	S	0.0	0.0	0:00.97	rcu_sched
8	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_bh
9	root	20	0	0	0	0	R	0.0	0.0	0:00.61	rcuos/0
10	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcuob/0
11	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration/0

Figure 9.8: Fedora top screenshot

The top program provides a dynamic real-time view of a running system. It can display system summary information as well as a list of tasks currently being managed by the Linux kernel. When the screen comes up there is a lot of data present and at first it might not be clear what you are looking at. The main key you need to know is `q` which will quit and exit the top command (just like the less command.) The image below displays the system average loads over longer rolling periods. 1 minute, 5 minutes, and 15 minute rolling average.

```
top - 13:57:27 up 58 min, 2 users, load average: 0.01, 0.09, 0.12
Tasks: 175 total, 2 running, 173 sleeping, 0 stopped, 0 zombie
%Cpu(s): 6.4 us, 0.7 sy, 0.0 ni, 92.9 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 2049520 total, 159148 free, 853616 used, 1036756 buff/cache
KiB Swap: 1888252 total, 1887768 free, 484 used. 996520 avail Mem
```

Figure 9.9: top avg

This section tells you the number of processes, how much memory and swap is in use and how much is free. It also tells you the breakdown between users and system on who is using the CPU percentage wise.

Finally this section shows the name and individual breakdown of the processes and how much resources they are using. We will cover this in more detail in chapter 11.

The `top` command also has the ability to sort and modify its output while running.

¹³[https://en.wikipedia.org/wiki/Top_\(software\)](https://en.wikipedia.org/wiki/Top_(software))

¹⁴<http://www.rogerbinns.com/blog/linux-top-origins.html>

```

top - 13:57:27 up 58 min, 2 users, load average: 0.01, 0.09, 0.12
Tasks: 175 total, 2 running, 173 sleeping, 0 stopped, 0 zombie
%Cpu(s): 6.4 us, 0.7 sy, 0.0 ni, 92.9 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem: 2049520 total, 159148 free, 853616 used, 1036756 buff/cache
KiB Swap: 1888252 total, 1887768 free, 484 used. 996520 avail Mem

  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
 1776 control+  20   0 1531748 202716 74984 S  3.3   9.9   2:21.08 gnome-shell
 1595 root        20   0 286592  71384 31044 S  3.0   3.5   0:42.81 Xorg

```

Figure 9.10: *top usage*

```

  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
 1776 control+  20   0 1531748 202716 74984 S  3.3   9.9   2:21.08 gnome-shell
 1595 root        20   0 286592  71384 31044 S  3.0   3.5   0:42.81 Xorg
 2348 control+  20   0 613512  34720 26156 S  1.7   1.7   0:12.02 gnome-terminal-
 3381 control+  20   0 1054136 225356 89436 S  1.0  11.0   0:30.24 firefox
 3578 root        20   0      0      0      0 S  0.3   0.0   0:00.10 kworker/0:0
 3672 control+  20   0 159820   4504  3804 R  0.3   0.2   0:00.03 top
 1 root        20   0 129048  8976  4144 S  0.0   0.4   0:02.62 systemd
 2 root        20   0      0      0      0 S  0.0   0.0   0:00.00 kthreadd
 3 root        20   0      0      0      0 S  0.0   0.0   0:00.28 ksoftirqd/0
 5 root        0  -20      0      0      0 S  0.0   0.0   0:00.00 kworker/0:0H
 7 root        20   0      0      0      0 S  0.0   0.0   0:00.97 rcu_sched
 8 root        20   0      0      0      0 S  0.0   0.0   0:00.00 rcu_bh
 9 root        20   0      0      0      0 R  0.0   0.0   0:00.61 rcuos/0
10 root        20   0      0      0      0 S  0.0   0.0   0:00.00 rcuob/0
11 root        rt   0      0      0      0 S  0.0   0.0   0:00.00 migration/0

```

Figure 9.11: *top processes*

Key	Action in Top
'd' or 's'	Plus a positive number you can change the reporting cycle.
'u'	Plus a user's name will filter only those processes they own
'k'	Sorts by %CPU usage.
't'	Sorts by CPU time usage
'm'	Same as above but more granular
'n'	Percentage of memory that a task is using.
'w' STATE	D=uninterruptible sleep, R=running, S=sleeping, T=traced or stopped, Z=zombie

9.5.2 htop

The htop command is an extension to the Linux top command. It is written in C using the ncurses library for text-based GUIs so it has mouse support. It also has metered output and uses all the same interactive commands as top. The homepage for the project can be found at <http://hisham.hm/htop>. The htop command needs to be installed via apt-get or yum/dnf.

```

CPU [||||| 76.5%] Tasks: 121, 252 thr; 2 running
Mem [||||| 939/2661MB] Load average: 0.02 0.10 0.13
Swp [||||| 0/1043MB] Uptime: 00:58:16

  PID USER      PRI  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  Command
 1776 control+  20   0 1495M 197M 74984 S  6.2   9.9   2:19.83 /usr/bin/gnome-shell
 1595 root        20   0 279M 71384 31044 S  1.9   3.5   0:42.46 /usr/libexec/Xorg vt2 -displayfd 3 -auth /r
 2348 control+  20   0 599M 34720 26156 S  1.4   1.7   0:11.90 /usr/libexec/gnome-terminal-server
 3664 control+  20   0 121M 3668 2964 R  1.0   0.2   0:00.07 htop
 1 root        20   0 126M 8976 4144 S  0.0   0.4   0:02.62 /usr/lib/systemd/systemd --switched-root --
 466 root        20   0 45876 13996 13424 S  0.0   0.7   0:01.78 /usr/lib/systemd/systemd-journald
 485 root        20   0 126M 8244 3096 S  0.0   0.3   0:00.01 /usr/sbin/lvmetad -f
 588 root        20   0 44212 4536 3188 S  0.0   0.2   0:00.26 /usr/lib/systemd/systemd-udev
 601 root        16  -4 49088 3124 2768 S  0.0   0.2   0:00.01 /sbin/auditd -n
 594 root        16  -4 49088 3124 2768 S  0.0   0.2   0:00.06 /sbin/auditd -n
 607 root        12  -8 80224 1668 1604 S  0.0   0.1   0:00.01 /sbin/audispd
 602 root        12  -8 80224 1668 1604 S  0.0   0.1   0:00.04 /sbin/audispd
 605 root        39  19 16784 2688 2436 S  0.0   0.1   0:00.00 /usr/sbin/alsactl -s -n 19 -c -E ALSA_CONFI
 606 root        16  -4 21984 2112 1940 S  0.0   0.1   0:00.01 /usr/sbin/sedispach
 640 rtkit       20   0 160M 2320 2124 S  0.0   0.1   0:00.03 /usr/libexec/rtkit-daemon
F1 Help F2 Setup F3 Search F4 Filter F5 Tree F6 Sort By F7 Refresh F8 Nice F9 Quit F10 Exit

```

Figure 9.12: *htop*

9.5.3 systemd-cgtop

You were probably wondering if systemd had its own system monitoring tool. And you would be correct to think so. Its name is entered atop and the command is native to our system running systemd. The usage patterns are

```

ATOP - localhost          2015/11/02 13:55:08          -----          10s elapsed
PRC | sys  0.58s | user  2.90s | #proc 175 | #tslpu 0 | #zombie 0 | no_procacct |
CPU | sys  6%   | user  29%   | irq    0% | idle   65% | wait    0% | curscal  7% |
CPL | avgl 0.08 | avg5  0.14 | avg15 0.14 | csw 10717 | intr  4816 | numcpu  1 |
MEM | tot  2.0G | free 154.7M | cache 829.7M | buff 76.2M | slab 105.3M | hptot  0.0M |
SWP | tot  1.0G | free  1.0G |          |          | vmcom  3.7G | vmlin  2.8G |
LVM | fedora-root | busy 1% | read  1 | write 133 | MBw/s 0.10 | avio 1.10 ms |
DSK |          | busy 1% | read  1 | write  94 | MBw/s 0.10 | avio 1.55 ms |

  PID  TID  SYSCPU  USRCPU  VGR0W  RGR0W  RUID  EUID  THR  ST  EXC  S  CPU  CMD  1/1
1776 - 0.12s  2.01s  0K  12K  controll  controll  7 -- - S 22% gnome-shell
1595 - 0.19s  0.59s  0K  0K  controll  root  1 -- - S 8% Xorg
2348 - 0.01s  0.18s  256K  248K  controll  controll  4 -- - S 2% gnome-terminal
795 - 0.15s  0.00s  32K  44K  root  root  1 -- - S 2% crond
3655 - 0.08s  0.03s  2640K  1892K  controll  controll  1 -- - R 1% atop
3381 - 0.01s  0.08s  0K  0K  controll  controll  46 -- - S 1% firefox
1310 - 0.00s  0.01s  0K  0K  gdm  gdm  7 -- - S 0% gnome-shell
7 - 0.01s  0.00s  0K  0K  root  root  1 -- - S 0% rcu_sched
3578 - 0.01s  0.00s  0K  0K  root  root  1 -- - S 0% kworker/0:0
1389 - 0.00s  0.00s  0K  0K  gdm  gdm  4 -- - S 0% goa-daemon
1 - 0.00s  0.00s  0K  0K  root  root  1 -- - S 0% systemd

```

Figure 9.13: *atop*

Command	Function
<code>free</code>	Report the amount of free and used memory in the system
<code>load</code>	Graphical representation of system load average
<code>uptime</code>	Display how long the system has been running
<code>vmstat</code>	Report virtual memory statistics
<code>w</code>	Report logged in users and what they are doing
<code>watch</code>	Execute a program periodically, showing output fullscreen

9.5.5.1 Load Generators

The opposite side of system monitoring is sometime you want to generate a load to see how your system responds. On modern system that are multi-core with fast memory. There is a tool called **stress** you can install it via The Ubuntu and Fedora software stores or form the commandline using `apt-get` and `yum/dnf`. It is also available for the Mac via the Homebrew package manager. The command `stress --cpu 2 --timeout 60` will cause two processors to max out for 60 seconds. You would then be able to see this using an of the above top based commands. There are some other ways to generate loads in bash as well located here: <http://stackoverflow.com/questions/7908953/how-to-measure-cpu-usage/12993326#12993326>

9.5.5.2 sar and iostat

In addition to memory, CPU, and process information. You can other commands to measure system I/O. The **sar** command - system activity report – is something that came from the BSD Unixes and was ported over to Linux. It is used to sample and report various cumulative statistic counters maintained by the operating system. You can take *n* samples at *t* intervals. Finally the **iostat** command, which in Linux is part of the `sysstat` package–displays kernel I/O statistics on terminal, device and cpu operations. A common word for this is I/O measurement or throughput. These reports can then be used to change system configurations to better balance the input/output load between physical disks.

```

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           1.77    0.02    0.37    0.65    0.00   97.19

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda                 1.99         37.93         25.15     707234     468853
dm-0                 0.01          0.06          0.00         1156          0
dm-1                 2.36         37.71         25.15     703061     468844

```

Figure 9.14: *iostat*

9.5.5.3 Procmon

Process Monitor (Procmon) is a Linux reimaging of the classic Procmon tool from the Sysinternals suite of tools for Windows. Procmon provides a convenient and efficient way for Linux developers to trace the syscall activity on the system.

9.5.5.4 iftop

to do three operations. The commands are: `useradd`, `userdel`, and `usermod`.

9.6.1 useradd

The `useradd` command allows you to add a new user to the system. You can set user defaults by typing the command below. The `-D` option will show you what your system `useradd` command defaults are. You can then add a user with these default values by typing:

```
useradd name-of-account-to-add
```

```
[controller@localhost default]$ cat useradd
# useradd defaults file
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
```

Figure 9.15: `useradd`

You have the option as well to override the default values and set your own values for a new user.

Table 9.5: `useradd` command options

Option	Function
<code>-c</code>	Add a comment
<code>-d</code>	User's home directory
<code>-e</code>	Date in which a user account expires
<code>-g</code>	Add user to a specific primary group
<code>-G</code>	Add user to additional supplementary groups
<code>-m</code>	Create user's home directory if it doesn't exist
<code>-s</code>	Assign the user's shell

```
controller@controller-VirtualBox:~$ sudo adduser joe
[sudo] password for controller:
Adding user `joe' ...
Adding new group `joe' (1001) ...
Adding new user `joe' (1001) with group `joe' ...
Creating home directory `/home/joe' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for joe
Enter the new value, or press ENTER for the default
  Full Name []: Joe
  Room Number []: Schmoie
  Work Phone []:
  Home Phone []:
  Other []:
Is the information correct? [Y/n] y
```

Figure 9.16: `adduser`

Example Usage: What do the options and arguments below do? Type it in and see what happens.

```
sudo useradd -c "This is a user for ITMO-356-01 Fall 2022" -d /home/controller \  
-G sudo -s /bin/ksh -m controller
```

In Debian distributions there is an abstraction layer called `adduser` and `addgroup` which are interfaces to the `useradd` and `groupadd` commands. It is just a perl script that passes the values you enter in the menu to the `useradd` command. On all other non-Debian distros `adduser` is a symlink to `useradd` command. The `adduser` command prompts you for information to fill out all the values and is recommended on Debian based systems, but if writing a shell script this is not portable to a non-Debian based distro.

9.6.2 userdel and usermod

The same as above, the `userdel` command allows you to delete a user. The `usermod` command allows you to modify a setting for a user without having to delete and re-create a user. The most common scenario is changing the users supplementary groups so that they can be in the `sudo`, `wheel`, or `admin` group. By default the system creates a `usergroup` with the same name as the `usergroup` and marks that as your users primary group. In this command `-a` means append and `-G` means append to the groups list. For example: `sudo usermod -aG sudo <username>` is a handy command to remember.

9.6.3 addgroup and groupadd

In the same way as a user is created and there is a Debian based shortcut, there is a similar command to create a new group. You would want to do this if you needed to create a group that was not in existence. The syntax is simply `sudo addgroup name-of-group` and that is it. The opposite command exists as well `groupdel` and `delgroup`, with the syntax of `sudo delgroup name-of-group`.

You can list all the groups that exist on your system by executing the `groups` command. If executed without any arguments it will show the current users group membership. If you follow it up with a username it will return the group memberships of that user. You can display the list of all the groups that exist on a system by typing `cat /etc/group` on the commandline.

9.6.4 The groups command

Sometimes you need to find detailed information about a user and which groups they are a member of. You can quickly inspect your current user by typing the command `groups` you will see print out a list of groups.

9.6.5 /etc/passwd

When a new user is created, the information passed into the `adduser` or `useradd` command is stored in the `/etc/passwd` file (yes it is missing the ‘or’). This file originally stored user passwords which had been encrypted, but the file had read access and it was realized that it was a security flaw to allow access in this way. The actual encrypted passwords were moved to a file called `/etc/shadow` and linked via the character `x` in the `/etc/passwd` file. You can see this in the image below. Also notice from the snippet that there are many many usernames that have been created but only two of them are by your hand. That is because the system upon install creates many additional users that have single or even legacy purposes that the user will not touch. At the very end of the screenshot below you see a user named `controller`, `vboxadd`, and `joe`. Two of those I created, the `vboxadd` was entered when I installed the VirtualBox Guest Additions.



```
speech-dispatcher:x:110:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/sh
avahi:x:111:117:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false
lightdm:x:112:118:Light Display Manager:/var/lib/lightdm:/bin/false
colord:x:113:121:colord colour management daemon,,,:/var/lib/colord:/bin/false
hplip:x:114:7:HPLIP system user,,,:/var/run/hplip:/bin/false
pulse:x:115:122:PulseAudio daemon,,,:/var/run/pulse:/bin/false
controller:x:1000:1000:controller,,,:/home/controller:/bin/bash
vboxadd:x:999:1:/var/run/vboxadd:/bin/false
joe:x:1001:1001:,,,:/home/joe:/bin/bash
```

Figure 9.17: `/etc/passwd`

The syntax is as follows: `Username:password:user-id:group-id:comment-field:home-directory:default-shell`. You can see the encrypted and salted password hash if you have root or `sudo` privileges by typing `sudo cat /etc/shadow` on the command line.

9.6.6 chmod

Pronounced “*chuh-mod*”. This command allows you to change the permissions or mode of a file. You can use numeric values to change the permissions all at once. Or you can use short cuts to assign or remove single permissions. The outputs look like this:

```
[controller@localhost Documents]$ ls -l
total 0
-rw-rw-r--. 1 controller controller 0 Sep 13 06:00 nightly-backup-script.sh
```

Figure 9.18: *Standard file permissions are 644 - very conservative and secure*

Why would you want to change permissions? You need to give write or execute permission to a file. Or to make sure that only users of a certain group can have write access (therefore delete access) to the content of a file. Or to give a shell script execute permission so it can be run by other.

```
[controller@localhost Documents]$ chmod 755 nightly-backup-script.sh
[controller@localhost Documents]$ ls -l
total 0
-rwxr-xr-x. 1 controller controller 0 Sep 13 06:00 nightly-backup-script.sh
```

Figure 9.19: *Same file with write and execute permission enabled*

9.6.7 chown

Pronounced “*chuh-own*”. This command allows you to change the owner of a file. The syntax would be `sudo chown root todo-list`. There is also a shorthand feature that allows you to change the group and the owner at the same time. `sudo chown root:root todo-list` the value following the semi-colon will be the new group owner. Resist the temptation to go nuclear—if you are getting a permissions denied on a command figure out why it is being denied—instead of chown-ing everything as this could create subtler problems down the line for system processes expecting a certain pattern of ownership.

Exercise: based on the previous todo-list.txt created in /tmp, issue an `ls -l` command - who is the owner of the file? Who is the group owner? Change it so that the file is owned by root and the group owner is root (remember to use `sudo`.)

9.6.8 chgrp

Pronounced “*Chuh-gerp*”. This is the change group command. It works just like `chown` but instead only changes the group ownership.

9.6.8.1 ACLs

If you have ever worked on Windows OS you will notice that they have much deeper access control and permission system than the basic read, write, execute and owner, group, other permissions. These are called ACL’s (pronounced “*ack-els*”) **Access Control Lists**. They are not native to the Linux world as they were not part of the original Unix standard. Modern versions of RHEL implement their own layer of Windows like ACLs on top of the regular permissions. The difficulties of ACL’s in Linux is that they are exclusive to RHEL and not portable to other Linux or Unix systems.

9.6.9 The 3 P’s of Troubleshooting Linux Problems

All my troubleshooting experience in Linux boils down to three things. I have named them the 3P’s (yes I know that they all don’t start with *P*). If you have an error message or cannot execute a command—run down these three troubleshooting steps:

- Path
 - If you get an error message telling you that `file not found` or `path does not exist` double check your path. Is the absolute path correct? Is it a relative path problem? Are you on the wrong level?
- Permission
 - Every file has permission on what is allowed to be done with it based on a simple access control of read write and execute. Maybe you don’t have permission to write and therefore can’t delete a file. Perhaps the file is owned by someone else and they didn’t give you permission. Check permissions via `ls -la` or see if you need `sudo`.
- dePendencies

- Are all the correct software dependencies installed? Perhaps you are missing a library or have an incompatible version that is preventing a tool from running? For example in the sample above running `runwhen`, you need Python3 installed. If you typed `python runwhen.py` you would receive a strange python error which would take you off on a useless googling experience? The problem is you needed to type `python3 runwhen.py` and if you don't have python3 installed, you will have a dependency problem.
- All else fails and you still have a problem, see if it is a full moon outside.

9.7 Secure Shell

What happens when you need to remotely access a system and it needs to be secure? The need for security or encryption of data sent over a network was not apparent. But as the ability to access data grew and the need to remotely access systems across untrusted networks became a reality the `rsh` remote shell was no longer viable. SSH or Secure Shell became a reality in 1999, appearing first in OpenBSD 2.6, introduced by the security focused OpenBSD project and quickly adopted universally across Unix, Linux, Mac, and now even Microsoft Windows. In fact [Microsoft was the first ever gold-level sponsor of the OpenBSD project](#).

By default the SSH *client* is installed on all Linux and Unix systems. This connection is authenticated via a username and password matching an account on the remote system or an RSA key.

You can access SSH from the command line via typing: `ssh -V`

```
[controller@localhost log]$ ssh -V
OpenSSH_8.0p1, OpenSSL 1.1.1c FIPS 28 May 2019
```

Figure 9.20: *Fedora Native ssh -V*

```
controller@controller-VirtualBox:~$ ssh -V
OpenSSH_7.6p1 Ubuntu-4ubuntu0.3, OpenSSL 1.0.2n 7 Dec 2017
controller@controller-VirtualBox:~$
```

Figure 9.21: *Ubuntu Native ssh -V*

`sudo apt-get install openssh-server` or `sudo dnf install openssh-server`

9.7.1 RSA keys

SSH works because of Public/Private Key Encryption and a standard created and widely adopted by the RSA company. Without going to deep into RSA encryption, this set of public and private keys allows you to securely transmit information across an untrusted network. How does it work?

Each person generates a **keypair**, a public key and a private key. Both halves of the key make up the single key used for authentication. These keys are exactly what they sound like. The public key is something that is revealed openly, but without the unique private key the “lock” cannot be opened. Think of the **public key** as the lock on your front door. Conceivably anyone can come up to that lock and try to insert a key. Unless they have the particular key, the lock won't open. The **private key** is then something to be guarded with your life as anyone who has that key can log into any system where it has permission.

How do you then exchange data? First you generate a keypair. On the command line you can issue the command: `ssh-keygen` and take notice of the prompts:


```

vagrant@ubuntu-xenial:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vagrant/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/vagrant/.ssh/id_rsa.
Your public key has been saved in /home/vagrant/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:FU47Y84Qzb3uiCtPsVaYGp/TiNW58VClWwfQWnBByfU vagrant@ubuntu-xenial
The key's randomart image is:
+---[RSA 2048]---+
|      .oo.  B*O|
|     +oo.= *+|
|    . B +. oE|
|    X =. .|
|   . S O.|
|  * B =.|
| o O.oo.|
| .O... .|
| oo.|
+-----[SHA256]-----+
vagrant@ubuntu-xenial:~$ ls ~/.ssh
authorized_keys  id_rsa  id_rsa.pub
vagrant@ubuntu-xenial:~$ █

```

Figure 9.22: *ssh-keygen* command output

9.7.1.1 SSH Security

While having SSH give us secure remote tunnels, it does lead to a potential problem. It means exposing an open port to the external network. This can and should be mitigated by things such as VPNs and mandating use of RSA keys only. But there are many systems that are exposed. This is a serious security vulnerability as hackers are actively scanning the entire IPv4 space looking for SSH systems and they will simply try to brute force the username and password.

One of the tools to alleviate this is called [fail2ban](#). This is a brute force login denial tool. This tool will scan the connection (or auth) logs looking for failed connection. Fail2ban can use the default syslog location as well as `journalctl` logs. Fail2ban will count the number of occurrences and the distinct IP and after a user defined threshold of failure will ban any network connection from the offending IP. This can be a time based back-off or can be a permanent ban, configured by the user in the configuration file. Fail2ban can also ban failed MySQL database connections as well if you have an exposed database server.

9.7.1.2 OpenSSL

OpenSSL is an OpenSource Library used for cryptographic key generation by OpenSSH. In 2016, it suffered an exploit due to the quality of the library maintaining older code from non-existent systems as well as being woefully underfunded and understaffed. Take note that although Google built its entire business using opensource and OpenSSL, they contributed almost nothing to its development. After the exploit a huge infusion of cash and adoption by the Linux Foundation of this project as a core infrastructure project has increased the quality of its security and development.

The heartbleed OpenSSL bug even has its own website to explain the details of it, located at <http://heartbleed.com>.

“The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. SSL/TLS provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs)¹⁵.”

“The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users¹⁶.”

Links to Security Now Technical Podcast explaining HeartBleed

- <https://twit.tv/shows/security-now/episodes/450> - Part 1
- <https://twit.tv/shows/security-now/episodes/451> - Part 2

“SAN FRANCISCO, April 24, 2014 – The Linux Foundation today announced it has formed a new project to fund and support critical elements of the global information infrastructure. The Core Infrastructure Initiative enables technology companies to collaboratively identify and fund open source projects that are in need of assistance, while allowing the developers to continue their work under the community norms that have made open source so successful. Founding backers of the Initiative include Amazon Web Services, Cisco, Dell, Facebook, Fujitsu, Google, IBM, Intel, Microsoft, NetApp, Rackspace, VMware and The Linux Foundation¹⁷.”

9.7.1.3 LibreSSL

Not to be outdone, the OpenBSD group immediately after HeartBleed, made a fork of the OpenSSL project code and called it [LibreSSL](#) and began to strip out as much unneeded code as possible and to write new code. This library became the default cryptographic library on OpenBSD and was ported to Linux. Though it is new and has a smaller attack surface, so many products (for better or worse) are using OpenSSL that they have not or cannot switch libraries. Microsoft adopted LibreSSL for use in their native SSH client and server in Windows 10 as well.

¹⁵<http://heartbleed.com>

¹⁶<http://heartbleed.com>

¹⁷<https://www.linuxfoundation.org/press-release/2014/04/amazon-web-services-cisco-dell-facebook-fujitsu-google-ibm-intel-microsoft-netapp-rackspace-vmware-and-the-linux-foundation-form-new-initiative-to-support-critical-open-source-projects/>

“LibreSSL is a version of the TLS/crypto stack forked from OpenSSL in 2014, with goals of modernizing the codebase, improving security, and applying best practice development processes.

Primary development occurs inside the OpenBSD source tree with the usual care the project is known for. On a regular basis the code is re-packaged for portable use by other operating systems (Linux, FreeBSD, Windows, etc)¹⁸.”

9.7.2 SFTP

Secure FTP uses the traditional FTP program but over a secure SSH tunnel. This allows you keep using existing file transfer methodologies but in a secure manner. FTP (file transfer protocol) is an unencrypted way to transfer files to and from a server. Its usage is discouraged as the protocol was developed at a time when security was not a consideration. All data, including passwords are transmitted in clear text. SFTP solves that issue of allowing you to use FTP but over an established SSH connection—there by using an SSH tunnel to provide encryption for the transmitted packets. Some would argue the rise in using version control such as Git makes SFTP/FTP redundant.

```
sftp [-i identity_file] username@hostname
```

9.7.3 SCP

Secure cp (copy) Allows for using the `cp` command to a remote system via SSH, as SFTP should be used for moving multiple files, this command is good for moving a single file quickly via the command line.

```
scp [-i identity_file] localfilename username@hostname:filename
```

9.7.4 Rsync

From the rsync man page:

Rsync is a fast and extraordinarily versatile file copying tool. It can copy locally, to/from another host over any remote shell, or to/from a remote rsync daemon. It offers a large number of options that control every aspect of its behavior and permit very flexible specification of the set of files to be copied. It is famous for its delta-transfer algorithm, which reduces the amount of data sent over the network by sending only the differences between the source files and the existing files in the destination. Rsync is widely used for backups and mirroring and as an improved copy command for everyday use.

9.7.5 ssh-copy-id

After generating an SSH keypair with the command `ssh-keygen`, you now have the two keys located in you `~/.ssh` directory. The file with the `.pub` extension is the public key, the other is the private key. **Guard the private key with your life!**

There is a command that will let you securely exchange RSA keys with a server. This requires you first to have an account on the server (host) you are connecting to so that you can place your identity securely onto that remote server.

```
ssh-copy-id username@hostname
```

```
# Optional command if you want to transfer the identity other than the default id_rsa  
ssh-copy-id -i identityname username@hostname
```

9.7.6 SSH config file

The `ssh` command has a provision for a file named `config` located in the `~/.ssh` directory. This is where you can hardcode short cut information per connection. Items such as:

- Turning off strict host key checking
- predefining hostname
- defining a non-default connection port
- predefining a password (not recommended)
- defining a non-default RSA private key

¹⁸<https://www.libressl.org/>

The format of the file is as such:

```
Host github.com
  User jhajak
  Hostname github.com
  IdentityFile /home/user/.ssh/id_ed25519_sample_student
```

The full range of options for the config file can be found in this Digital Ocean Tutorial located at <https://www.digitalocean.com/community/tutorials/how-to-configure-custom-connection-options-for-your-ssh-client>

9.7.7 SSH Service Daemons and Security

SSH has two configuration files that control its abilities. Located at `/etc/ssh/ssh_config` and `/etc/ssh/sshd_config`. The first file contains user information about what options your client will present and use when connecting to a remote SSH server.

In the `ssh_config` file you can modify these lines to increase the security of the encryption ciphers you use. By default OpenSSH defaults back to an older and weaker set of encryption ciphers such as:

```
Ciphers aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128,aes128-cbc,3des-cbc
```

You would uncomment the line and the entries with a more robust list. You would want to make sure that the corresponding SSH server in the `sshd_config` file had the same ciphers set otherwise negotiation could fail and no connection would take place¹⁹.

```
# This is one line with no line breaks
# I made it into two lines so it would display properly.
Ciphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,
aes128-gcm@openssh.com,aes256-ctr,aes192-ctr,aes128-ctr
```

You could also disable password authentication to a remote server and only use RSA key authentication. Uncomment the line `#PasswordAuthentication yes` and change it to `PasswordAuthentication no`²⁰. By default Fedora and BSD based operating systems allow the Root account to connect via SSH using a password. This is inherently dangerous, Ubuntu uses the value `prohibit-password` which would allow RSA but not password based auth. It is generally not a good idea to allow this and uncomment and change this setting from `PermitRootLogin yes` to `PermitRootLogin no`²¹.

Upon changing any values you need to inform `systemd` that a config file changed, it will find it and process it: `sudo systemctl daemon-reload`.

9.7.8 WireGuard - Linux kernel native VPN

A [VPN](#) is a virtual private network. It uses RSA encryption to extend a private network across the public network. An example would be an employee connecting from home over the public network to a company's private network. To do this you would need a VPN.

[WireGuard](#) is a Free and OpenSource VPN software. It's modularly built and since March of 2020, has been included natively in the Linux Kernel. This increases speed and flexibility compared to other VPN solutions. It was written by Jason A. Donenfeld and is published under the GNU General Public License (GPL) version 2, which makes it compatible with the Linux Kernel as well as being compiled for [MacOS](#), [Windows](#), [Android](#), [iOS](#), [FreeBSD](#), and [OpenBSD](#). Of note, launched on July 15th, Mozilla launched a VPN service, called [Mozilla VPN](#) that runs the wireguard code.

9.8 Chapter Conclusions and Review

Through this chapter we learned about the `su`, `sudo`, and root user account paradigms. We learned when to use them and how they were designed. We learned about the nature of traditional logging (non-`systemd`) and how they are

¹⁹<https://unix.stackexchange.com/questions/333728/ssh-how-to-disable-weak-ciphers>

²⁰<https://askubuntu.com/questions/2271/how-to-harden-an-ssh-server/2279>

²¹<https://askubuntu.com/questions/449364/what-does-without-password-mean-in-sshd-config-file>

stored. We learned about a newer logging format in the journald service from systemd. Finally we learned about system monitoring tools for visual display of system resources being used. Finally we learned about the 3Ps of Linux troubleshooting.

9.8.1 Review Questions

- 1) What user account has superuser privilege in Linux?
 - a. `sudo`
 - b. `su`
 - c. `superuser`
 - d. `root`
- 2) Which command do you use to temporarily elevate your user's privilege to the superuser (root)?
 - a. `su`
 - b. `sudo`
 - c. `su -`
 - d. `root`
- 3) How can you display the content of a file named `topsecret.txt` that has permissions `000` and is owned by another user?
 - a. You can't do that
 - b. `root cat topsecret.txt`
 - c. `sudo cat topsecret.txt`
 - d. `su cat topsecret.txt`
- 4) What license is the `sudo` application under?
 - a. GPL
 - b. BSD
 - c. Public Domain
 - d. ISC
- 5) Which operating system doesn't have an active root account by default?
 - a. Debian
 - b. Ubuntu
 - c. All Debian based distros
 - d. Fedora
- 6) What is the name of the file where `sudo` privilege are kept?
 - a. `/etc/sudo`
 - b. `visudo`
 - c. `/etc/allow`
 - d. `/etc/sudoers`
- 7) What is the name of the command used to modify `/etc/sudoers` to grant a new user `sudo` privilege?
 - a. Just use `vi` to edit it directly
 - b. Logout and log back in as `root` and do it
 - c. `visudo`
 - d. `sudo visudo`
- 8) Based on this line in `/etc/sudoers` - `%meninblack ALL=(ALL:ALL) ALL` - what does the first value by the `%` mean?
 - a. Name of a group
 - b. Name of a user
 - c. Name of the user group
 - d. Name of a process
- 9) In the `/etc/sudoers` file - what does this line mean: `RMS ALL=(root) NOPASSWD: ALL`

- a. The user RMS has sudo permissions and access to all commands
 - b. The user RMS has sudo permissions
 - c. The group RMS has sudo permissions to all commands
 - d. The user RMS has sudo permissions and access to all commands, and requires no password to elevate to the sudo user
- 10) When using the su command to switch from a regular user account to the root user account, what do you type to return to the standard user account?
- a. quit
 - b. exit
 - c. stop
 - d. sudo reboot
- 11) What command would you use to edit the file at this location: /var/www/html/index.html?
- a. vi /var/www/html/index.html
 - b. sudo vim /var/www/html/index.html
 - c. vim /var/www/html/index.html
 - d. You need to **chown** the file and change the owner
- 12) On a Linux system, which directory are all the traditional system (non-systemd) logs kept in?
- a. /var/run
 - b. /logs
 - c. /var/adm/log
 - d. /var/log
- 13) Under systemd and journald where are the logs kept?
- a. /var/log
 - b. /var/log/error
 - c. /var/log/journald
 - d. Trick question - as logs are stored in a binary format and retrieved via journalctl
- 14) What is the command you use to query the system logs in systemd?
- a. systemctl
 - b. journald
 - c. journalctl
 - d. showlogs
- 15) How would you filter the systemd log based on time? (Which is valid syntax?)
- a. journalctl --since=yesterday
 - b. journalctl --since=tomorrow
 - c. journalctl --yesterday
 - d. journalctl --filter=yesterday
- 16) Where is the journald.conf file located?
- a. /etc/logrotate.conf
 - b. /etc/systemd/journalctl.conf
 - c. /etc/systemd.conf
 - d. /etc/systemd/journald.conf
- 17) What command provides a dynamic real-time view of a running system?
- a. top
 - b. iostat
 - c. systmed-top
 - d. journalctl
- 18) Debian based distros have an additional command to abstract the process to add users to the system - what is it?

- a. useradd
 - b. usermod
 - c. adduser
 - d. add
- 19) What command would be used to modify a user account settings and add them to the sudo users group on an Ub untu distro (user is named controller)?
- a. `sudo useradd -aG sudo controller`
 - b. `sudo usermod -aG sudo controller`
 - c. `sudo usermod -G sudo controller`
 - d. `sudo userdel controller`
- 20) Which below are valid useradd commands? (Choose all that apply)
- a. `sudo useradd -c "User for spring class" -d "/home/export/controller" -G sudo -s /bin/ksh -m controller`
 - b. `sudo useradd -D controller`
 - c. `sudo useradd controller`
 - d. `sudo useradd -G sudo -s /bin/ksh -m controller`
 - e. `sudo useradd -c "User for spring class" -G sudo -m controller`

9.8.2 Podcast Questions

DevOps https://www.redhat.com/en/command-line-heroes/season-1/devops-tear-down-that-wall#tab.show_info.1

- ~0:30 What do developers do and whose problem is it?
- ~0:54 What are the two opposing sides and why are they opposed to each other?
- ~1:00 Who is the host and who does she work for?
- ~1:20 For decades how was the IT world defined?
- ~3:10 These days, a company like Amazon will deploy new code how many times every minute?
- ~4:50 Where does Joana Horowitz work?
- ~5:20 What is one of the biggest ways to increase uptime for an application?
- ~5:41 What do developers commit to and what does operations commit to?
- ~6:25 Who coined the term DevOps and in what year?
- ~6:58 Who is Scott Hanselman?
- ~7:24 How does Scott define DevOps?
- ~8:24 DevOps is not just about Code release velocity, what is the other vital part of that code release?
- ~12:18 Why does a business write and deploy software?
- ~13:00 The Agile methodology says who should own the product? IT or the Developers?
- ~16:48 Is DevOps a mindset or a set of tools?
- ~18:25 Beyond IT and Developers, who else has to buy in for the DevOps mindset to take place?

9.8.3 Lab

9.8.3.1 Lab Objectives

The objective of this lab are as follows:

- Understand when and how to use the sudo command
- Understand how to edit the `/etc/sudoers` file
- Understand how to use the journald and journalctl logging mechanism in systemd
- Understand how to add and manage user accounts
- Understand how to modify, use, and secure the SSH service

9.8.3.2 Lab Outcomes

At the outcome of this lab you will be able to successfully understand how to apply the sudo/root user paradigm. You will understand the binary logging mechanism of journald. You will be able to add, delete, and modify user accounts. Finally, you will be able to successfully setup and establish remote connections.

Assumptions:

- All questions need to display the content of each answer as a shell script and also display the output of that shell script - unless stated otherwise
 - Assignments can be done using Ubuntu or Fedora desktop unless stated.
 - For a challenge try to use Ubuntu Server
 - If a command asks you to work on a user that doesn't exist, it is assumed that you have to create it.
 - The `mysqldump` application requires the `mysql-client` package to be installed: <http://superuser.com/questions/165582/installation-mysqldump>.
- 1) Write a shell script that issues the command to add a user named “controller” to your system, using the system default values and display the content of the `/etc/passwd` file to show that the user has been created
 - 2) Write a shell script that issues the command to add a group named `itmo356` to your system. Then use the command to modify the user and append this additional group to the user controller you just created. Issue the command that shows all of the groups a user is a member of.
 - 3) Write a shell script that issues the command to modify the user **controller** to add them to a `superuser` group (sudo on Ubuntu or wheel on Fedora based)
 - 4) Write a shell script that issues the command to add the user named **nsa**, display the content of the `/etc/passwd` file, and then delete the user account, display the content of the `/etc/passwd` to show that the user has been deleted
 - 5) Issue the commands to edit the `/etc/sudoers` file and give the **user** “mysql-backup” sudo privilege. Take a screenshot of the relevant line that you added to `/etc/sudoers` file.
 - 6) Issue the commands to edit the `/etc/sudoers` file and give the **group** “mysql-admins” sudo privilege. Take a screenshot of the relevant line that you added to `/etc/sudoers` file.
 - 7) Issues the commands to edit the `/etc/sudoers` file and give the **user** “mysql-admin” sudo privilege to only use the mysql database backup command named `mysqldump`. Take a screenshot of the relevant line that you added to `/etc/sudoers` file.
 - 8) Issue the commands to edit the `/etc/sudoers` file to give the user “mysql-admin” sudo privilege to only execute the `mysql` command and not require a password. Take a screenshot of the relevant line that you added to `/etc/sudoers` file.
 - 9) Issue the command to list all log messages of priority levels ERROR and worse, from the current boot.
 - 10) Issue the command to install the Nginx webserver from the command line using either `dnf` or `apt-get`.
 - 11) Issue the command to list all the occurrences in the logs generated by nginx using `journalctl`.
 - 12) In your Virtual Machine open a webbrowser and navigate to `http://127.0.0.1`. Then type in the address bar `http://127.0.0.1/cats` (this site will 404 – not be found). Then issue the command to display the content of the Nginx `access.log` showing both connection you just made, from the log file
 - 13) In your home directory, create a file named: `todo-list.txt`. Then create a group named: `accounting`. What command would you use to change the group ownership of the file `todo-list.txt` to be owned by the “accounting” group?
 - 14) Issue the command you would type to generate a RSA key pair, if you created it previously you can overwrite the previous key.
 - 15) The next questions require some setup:
 - i. You need two virtual machines for this part: One Ubuntu based and one Fedora based
 - ii. You will need to modify the Network settings to **Bridged** in VirtualBox to get a public IP (if you are at home your router should suffice, if you are on campus you can come to the lab).
 - iii. Install **openssh-server** on Ubuntu (Server) via `apt` and Fedora via `dnf`
 - iv. On Fedora only, you will need to issue two additional commands to start the ssh server: `sudo systemctl enable sshd` and `sudo systemctl start sshd`
 - 16) Clone the repository <https://github.com/arthepsy/ssh-audit> to both the client and server system, in your home directory. Run the `ssh-audit` tool on the Fedora and Ubuntu, list the weak ciphers installed by default

- 17) Modify the client and servers using the example in the text to increase cipher strength, run the ssh-audit tool again and report any weak ciphers or security anomalies.
- 18) Issue the command to connect remotely to the Fedora server via ssh using username and password, create a text file named: website.app - then exit the remote session. On the Fedora system show that the file: website.app, has been created
- 19) Issue the command to transfer an RSA key pair to from your system designated as a client, Fedora, to the Ubuntu SSH server
- 20) Issue the command to connect remotely to the Fedora server via ssh using username and the identifyFile (RSA key), create a text file named: new-website.app - then exit the remote session. On the Fedora system show that the file: website.app, has been created

9.8.3.3 Footnotes

Chapter 10

Init Services, Daemons, and Processes

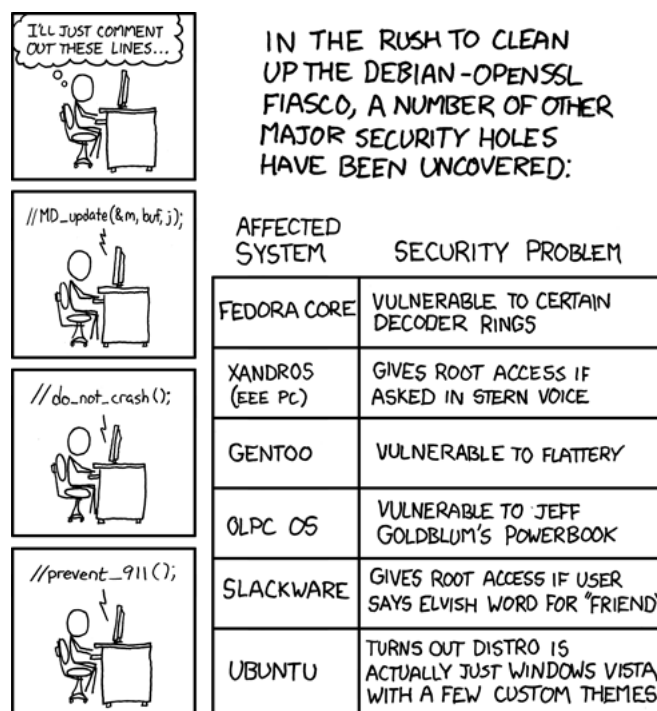


Figure 10.1: *You never know what is enabled...*

10.1 Objectives

- Understand the traditional SysVinit system and its relationship to starting system processes
- Understand the systemd init system
- Understand the nature of starting, stopping, and service reporting
- Understand how to use system tools to examine processes
- Understand the nature of the /proc virtual-filesystem

10.2 Outcomes

At the outcome of this chapter you will have an understanding of how the traditional init system and the new systemd init system comparatively work. You will understand how to examine, start, and stop services in both arenas. You will learn about the /proc virtual-filesystem and how it represents running processes as files. You will be able to use the systemd mechanisms for process reporting and termination.

10.3 First Phase of the system boot

10.3.1 BIOS

When you hit the power button, after a short pause, your system begins to whirl to life. Fans spinning and a short POST *beep* sound tells you are on your way. The next thing you see is some messages flash across the screen, a logo perhaps, some spinning icons, and then the login screen comes to life. Collectively this is called the *boot* process. We are interested in the last part—when the operating system is loaded from disk into memory and the init process begins to start launching the components of our operating system. Before we dive into that part, let’s review what goes on before hand.

How does the system come to life? There is a small chip on your motherboard that has some basic drivers hard coded into it. This chip loads these basic drivers into memory with just enough access to start to load the operating system from disk. Traditionally this was called the **BIOS**, Basic Input Output System. This BIOS stems from the first PC device The IBM PC which developed it back in 1979. When the BIOS boots and successfully receives a positive result from the POST test, which will be a short *beep*, there are hard coded basic I/O device drivers that are loaded into memory to allow the computer to complete basic I/O functions, display information, and read from a hard disk. Immediately the system loads the MBR—Master Boot Record, which is the first sector of the first identified bootable device. This is usually your hard drive but could be a flash drive, SD Card, or even a good old fashioned CD-ROM. The Master Boot Record points to the VBR or Volume Boot Record.

In modern Linux the VBR is controlled by **GNU GRUB**. *“Briefly, a boot loader is the first software program that runs when a computer starts. It is responsible for loading and transferring control to the operating system kernel software (such as the Hurd or Linux). The kernel, in turn, initializes the rest of the operating system (e.g. GNU).”*¹

10.3.2 UEFI

The **Unified Extensible Firmware Interface** was developed as a successor to the BIOS. The original BIOS spec only supported a 16-bit real-mode and 1 MB of memory, which limited what could be done on a system. The replacement UEFI, does the same job as BIOS but is a modern reimplementaion of BIOS’s basic functions. Who runs **UEFI**? It is controlled by a trade organization which all the major PC makes and OS vendors purchase a seat in the UEFI consortium.

UEFI capabilities include additional features not available in the traditional BIOS. Most modern Laptops and Servers, since 2015 are using UEFI, some come with a BIOS Compatibility Layer, but BIOS and UEFI are two separate things. When booting with UEFI, at stage 4 the UEFI Boot Manager entry that holds the location of the bootable device is loaded.

- 1) SEC - Security Phase
- 2) PEI - Pre-EFI Initialization
- 3) DXE - Driver Execution Environment
- 4) BDS - Boot Device Select
- 5) TSL - Transient System Load
- 6) RT - Runtime

UEFI supported is detected on install by all operating systems. You can specifically enable it in VirtualBox 6+:

10.4 Second Phase of the system boot

10.5 GNU GRUB Boot Loaders

GNU GRUB superseded what is now called legacy GRUB (version 0.9x) and begins with version 2 to separate the different tools—though they are both called GRUB. We will only talk here about GRUB 2 or GNU GRUB when referring to GRUB. GRUB itself has 3 stages in order to get you to stage two which is the loading of your Linux Kernel from disk into memory. Stage 1 detects and finds the locations of and discovers various file systems on disk for loading at a later time. Once this is accomplished GRUB loads stage 1.5 and passes control to it. GRUB 1.5 will load file system drivers that are separate from the Operating System so the file `/boot/grub/grub.cfg` can be read, Fedora stores its grub configuration at `/boot/grub2/grub.cfg`. This file contains details about the path to various

¹<https://www.gnu.org/software/grub/>

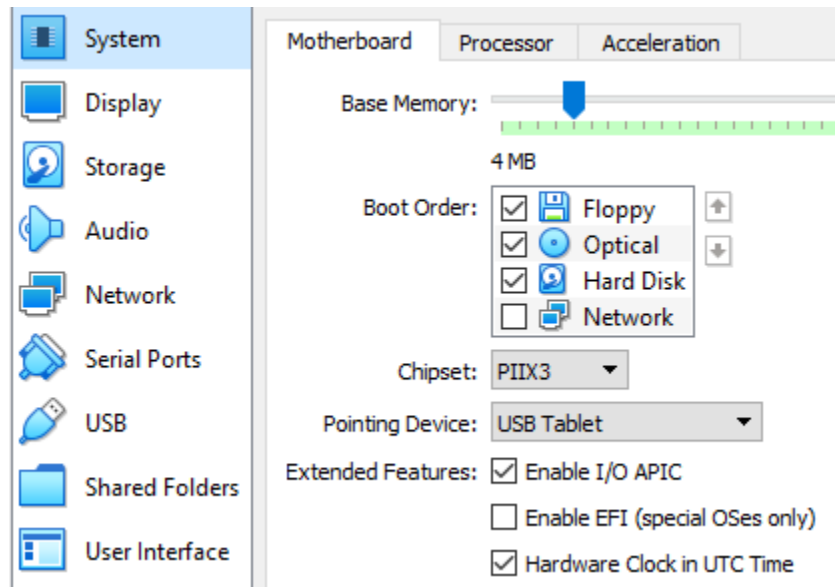


Figure 10.2: *Enable EFI*

kernels for loading and various system configurations. Once complete stage 2 is loaded and control is passed. This is where you get the nice TUI (Terminal User Interface screen) allowing you to choose which kernel and any features you want to enable/disable per this boot.

By default this User Interface will pop up if you have more than one operating system or kernel version installed in the case of Fedora. If you have a single operating system this screen will be skipped by default, you can hold down the SHIFT key at boot and force this screen to come up. In the case of Ubuntu you can select ADVANCED to see different kernel versions you can load.

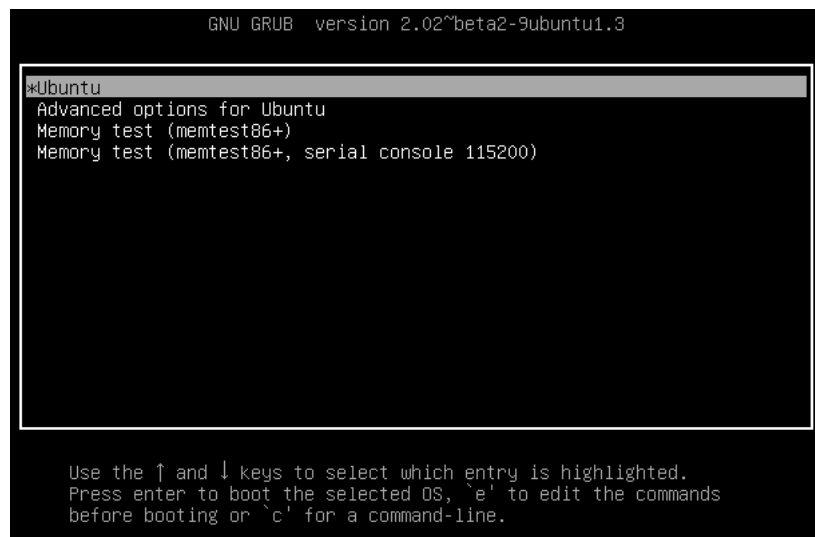


Figure 10.3: *Terminal User Interface*

Once we select a kernel version, GRUB knows where to go to find that kernel file, read it into memory, decompress it. All kernel images are located in `/boot` and your GRUB 2 configuration file knows this.

You will notice that there is a `vmlinuz` kernel image per each instance that corresponds to the TUI entries in the previous image. The file that is loaded first is actually the `initrd.img-X.XX.X-XX` file. This is the pre-kernel which contains all the drivers necessary for the kernel to use before the root filesystem has been mounted. The `initrd` file is gzip compressed and is decompressed on each boot. Once the `initrd` temporary filesystem is loaded, with its own `/etc` and own `/bin`, the `vmlinuz` file which is the actual kernel is now loaded into memory and begins to un-mount and

remove the *initrd* from memory as it is no longer needed.

10.5.1 GNU GRUB Settings

The GRUB 2 or GNU GRUB bootloader exists in the file `/boot/grub/grub.cfg` but this file is auto generated so to edit the settings you would modify the `/etc/default/grub` file.

The `/etc/default/grub` file contains various key, value pairs defining default kernel parameters to be passed to GRUB.

`GRUB_DEFAULT=N` – this value is which entry in your GRUB list is the default operating system to boot. If you have a single OS installed, this value will be 0.

`GRUB_TIMEOUT` – this is the setting that tells the system to boot the default command N seconds after the GRUB menu has appeared. It functions as a countdown.

`GRUB_TIMEOUT_STYLE` – this will let the count down appear or silently countdown.

`GRUB_CMDLINE_LINUX_DEFAULT` – this value adds additional key value pairs to the kernel boot parameters. A common setting here is to change the setting of *splash quiet* which hides and boot time kernel message behind the Ubuntu logo and instead displays them on the screen.

`GRUB_DISABLE_RECOVERY` – this hides the single user/recovery mode from the GRUB menu per kernel entry

`GRUB_GFXMODE=640x480` – this setting is commented out by default, but you can enable this to hard code a certain boot resolution.

`GRUB_BACKGROUND` – this option lets you *theme* your GRUB menu by adding a background image.

To make these changes permanent you need to execute the `sudo update-grub` command after saving the file so the `/boot/grub/grub.cfg` will be regenerated and used on the next boot.

10.6 Third Phase of the system boot

10.6.1 SysVinit

I am going to describe the Unix System V init process - this is the basis of all Unix and Linux knowledge since the early 1980s. This is referred to as SysVinit–note that only the Unix based derivatives of BSD use this as of 2018. SysVinit is not in use in Linux as it was replaced by systemd as the init system.

Once the kernel has complete control of the hardware, it begins to execute the “guts” of the operating system–by setting up the system processes. The first task it executes is `/sbin/init`. This is referred to as the init process. It’s job is only to be the ancestor of all other processes and start each succeeding service–starting from the X server, to the login server, to any GUI, to a webserver or database. The `/sbin/init/` looks at the value stored in `/etc/inittab` to find the system **run level**. The **run level** tells us which mode to start in and which associated services are needed. These levels are general and each Linux distro modifies them as needed. The default run level for a GUI based distro is **5**. The default run level for a server based OS is **3**.

Table 10.1: Traditional Run Levels

Level	Operation
0	System Halt/Shut Down
1	Single User Mode
2	Multuser Mode Without Networking
3	Full Multuser Mode
4	Unused
5	GUI/X11
6	Reboot

Once the run level is determined, there is a directory called `/etc/rc.d` which contains what are called **run level specific** programs to be executed. Files preceded by an *S* mean to start the service, and files preceded by a *K*

mean to kill that service. Each K or S file is followed by a number which also indicated priority order—lowest is first. The good thing is that each K or S file is nothing more than a bash script to start or kill a service and do a bit of environment preparation. As you can see this system has some flaws. There is no way to start services in parallel, its all sequential, which is a waste on today’s modern multi-core CPUs. Also there is no way for services that start later that depend on a previous service to be started to understand its own state. The service will happily start itself without its dependencies and go right off a cliff ².

```
vagrant@packer-virtualbox-iso-1418239407:/etc/init$ cd /etc/rc
rc0.d/ rc1.d/ rc2.d/ rc3.d/ rc4.d/ rc5.d/ rc6.d/ rcS.d/
vagrant@packer-virtualbox-iso-1418239407:/etc/init$ cd /etc/rc
.d/
vagrant@packer-virtualbox-iso-1418239407:/etc/rc3.d$ ls
README                S50saned              S99ondemand
S20kerneloops        S70dns-clean         S99rc.local
S20rsync              S70pppd-dns
S20speech-dispatcher S99grub-common
```

Figure 10.4: Classic SysVinit RC files on Ubuntu 14.04

10.6.2 Working With Services in SysVinit/Upstart

Under the Upstart methodology you can simply start services and stop them with the `service` command. The syntax is `sudo service <service-name> start | stop | restart | reload | status`. This would act upon the appropriate shell script to perform the appropriate action. Why would you need to restart a user run service? Remember that everything in Linux is configured with text files. At initial load the text files information is parsed and placed in memory. If you change a value, you need to reload that configuration file into memory, and restarting a service does just that for instance. The `service` commands are still in place but since Ubuntu 15.04 and Fedora 20 they are just symlinks to the `systemd` command and control `systemctl`.

Example Usage: On an Ubuntu system to restart your apache2 webserver your would type: `sudo service apache2 restart` (assuming you had apache2 already installed).

Example Usage: On SysVinit systems (pre-Ubuntu 6.10) you would type the absolute path to the directory where the init script was located. In this case perhaps `/etc/init.d/apache2 restart`

10.6.2.1 ps

The `ps` command displays information about a selection of the active processes. This is different from the `top` command as the information is not updated but just displayed. The `ps` command by itself shows very little useful information. Overtime three versions of ‘ps have joined together so there are three sets of options, BSD, Unix, and GNU. The BSD options have no “-” prefix, UNIX options have a single “-” and GNU options have a double dash “--”.

```
controller@controller-VirtualBox:~$ ps
  PID TTY          TIME CMD
 2063 pts/18    00:00:00 bash
 2073 pts/18    00:00:00 ps
```

Figure 10.5: ps command

These additional commands will share more information:

- `ps -e` <- select all processes (similar to -A)
- `ps -ef` <- this is one of the more helpful and verbose sets of options with full-formatting
- `ps -eF` <- Extra full-formatting
- `ps -ely` <- Long formatting

²<http://www.slashroot.in/linux-booting-process-step-step-tutorial-understanding-linux-boot-sequence>

- `ps -eo pif,tid,class,ni,pri,psr...` <- the `o` option allows you to customize the column arrangement and output.
- `ps -C syslogd -o pid=` <- this is the same as doing `ps -ef | grep firefox` or `pidof firefox`
- `ps xawf -eo pid,user,cgroup,args` ³ Shows `cgroup` ownership details.
- `systemd` version of `ps` is called `systemd-cgls` which shows a nice hierarchy of process ownership.
 - `cgroups` (control groups) were a feature added to the Linux kernel that allow for processes to be grouped together and control commands can be executed on entire groups (permission limiting, start/stop, priority changes, etc, etc.) `Systemd` makes big use of `cgroups`.

10.6.2.2 kill

In the SysVinit/Upstart world to terminate a process you would use the `kill` command. There are various levels of `kill`.

Level	Name	Function
1	SIGHUP	Used to make a process re-read a config file
2	SIGINT	Effectively hitting CTRL+C
9	SIGKILL	Kills a process ungracefully—could damage files, use as last resort
15	SIGTERM	Like a kill 9, but with class, gracefully kills a process

All programs can choose to *trap* these kill commands and ignore them or take different expected behaviors. All except `kill -9` every process has to obey. You can use the `killall` command to kill the process and any associated processes that it had launched all in one fell swoop. You can use the `ptkill` command to kill a process by name instead of PID.

10.6.2.3 nice

The `nice` command is a *suggestion* tool to the operating system scheduler on how to adjust resource allocation to a process. Giving `nice` the value or `-20` means that this is a really high priority or more favorable process, all the way up to 19 which means that it is a really low priority process. A good example of this would be on a large print job that will take a long time to print but you are not in a time rush—so you can `nice` the print job to a low priority and it will print when the system is less busy. You can find the usage at `man nice`.

Example Usage: This example will increase favorability of this process to the scheduler by 10 on a scale of `-20` to `19`—default is `0`.

```
nice -n 10 my-loop
```

10.6.3 Upstart

In 2006 the Ubuntu distro realized the short comings of SysVinit and created a compatible replacement called `upstart`. `Upstart` moved all the traditional runlevels and start up scripts to `/etc/init` directory and placed the scripts in configuration files. While leaving the `/etc/rc.d` structure in place for any backward compatible needing scripts. Here is an example of a `myservice.conf` `upstart` file stored in `/etc/init/myservice.conf`: Note the use of the **run level** concept from SysVinit. Compare this to (on an Ubuntu system) the contents of any script in `/etc/rc3.d` (run level 3). `Upstart` exhibits a bit more process control but still being a shell script when you boil it down.

```
myservice - myservice job file
description "my service description"
author "Me <myself@i.com>"
# Stanzas
#
# Stanzas control when and how a process is started and stopped
# See a list of stanzas here: http://upstart.ubuntu.com/wiki/Stanzas#respawn
# When to start the service
start on runlevel [2345]
```

³<http://0pointer.de/blog/projects/systemd-for-admins-2.html>

```

# When to stop the service
stop on runlevel [016]
# Automatically restart process if crashed
respawn
# Essentially lets upstart know the process will detach itself to the background
expect fork
# Run before process
pre-start script
    [ -d /var/run/myservice ] || mkdir -p /var/run/myservice
    echo "Put bash code here"
end script
# Start the process
exec myprocess

```

Ubuntu adopted Upstart in 2006, Fedora adopted it as a SysVinit supplemental replacement in Fedora 9 - until version 18 when systemd was ready. RHEL and CentOS used Upstart as well until RHEL 7, and ChromeOS still does (OS for Chromebooks). Debian considered using Upstart when Debian 8 was being developed but instead decided to jump entirely to systemd instead. When Debian made the jump, this forced Ubuntu, which is a Debian derived distribution, to abandon work on Upstart and switch to systemd as their init system—though they fought until the bitter end. Upstart was seen as the compromise between SysVinit and its failings but in the end systemd won out. MacOS uses their own version called [launchd](#) and Sun/Oracle Solaris and Illumos/SmartOS use [SMF](#) which are similar to Upstart in concept but have OS specific extensions.

10.6.4 Other SysVinit replacements

Upstart wasn't the only replacement option, currently there are two major ones, [OpenRC](#) and [runit](#). OpenRC is maintained by the Gentoo Linux developers, runit is focused on being “a cross-platform Unix init scheme with service supervision, a replacement for sysvinit, and other init schemes. It runs on GNU/Linux, BSD, MacOSX, Solaris, and can easily be adapted to other Unix operating systems⁴.”* Devuan Linux, which is the Debian fork without systemd, still uses sysVinit but has the ability to use OpenRC or runit if you so choose.

OpenRC and runit do not use systemd at all and therefore any software that requires systemd as a dependency, such as the [GNOME desktop](#), then cannot be used. These new projects maintain the backward compatibility of SysVinit but improve or adopt systemd style improvements and management where feasible. Here is a comparison table between systemd, sysVinit, and OpenRC:

systemd	SysVinit	OpenRC
<code>systemctl list-units</code>	<code>service --list-all</code>	<code>rc-status</code>
<code>systemctl --failed</code>	-	<code>rc-status --crashed</code>
<code>systemctl --all</code>	-	<code>rc-update -v show</code>
<code>systemctl start/stop</code>	<code>service start/stop daemon</code>	<code>rc-service daemon start/stop</code>
<code>systemctl enable/disable</code>	<code>chkconfig on/off daemon</code>	<code>rc-update add/del daemon</code>
<code>systemctl daemon-reload</code>	<code>chkconfig daemon -add</code>	-

Init Systems: Comparison of actions

10.6.5 Systemd and Systemctl

Systemd was the alternative decision to SysVinit/Upstart that had been developed by Lennart Poettering while at Red Hat. It's main goal is to unify basic Linux configurations and service behaviors across all distributions. Systemd is licensed under the LGPL 2.1 or later, [GNU Lesser General Public License](#).

From Lennart's own website, “*systemd is a suite of basic building blocks for a Linux system. It provides a system and service manager that runs as PID 1 and starts the rest of the system*⁵.” Unlike the SysVinit/Upstart method which has an ancestor process in PID 1 (process ID 1), now systemd becomes the PID 1. The init process *IS* the system

⁴<http://smarden.org/runit/>

⁵<http://www.freedesktop.org/wiki/Software/systemd/>

and the process manager, if PID 1 dies, then your system dies too. Systemd includes many other items, 69 different binaries all rolled into PID 1.

“As an integrated software suite, systemd replaces the startup sequences and runlevels controlled by the traditional init daemon, along with the shell scripts executed under its control. systemd also integrates many other services that are common on Linux systems by handling user logins, the system console, device hotplugging (see udev), scheduled execution (replacing cron), logging, hostnames and locales⁶.”

“One goal of systemd is to unify the dispersed Linux landscape a bit. We try to get rid of many of the more pointless differences of the various distributions in various areas of the core OS. As part of that we sometimes adopt schemes that were previously used by only one of the distributions and push it to a level where it’s the default of systemd, trying to gently push everybody towards the same set of basic configuration. This is never exclusive though, distributions can continue to deviate from that if they wish, however, if they end-up using the well-supported default their work becomes much easier and they might gain a feature or two. Now, as it turns out, more frequently than not we actually adopted schemes that where Debianisms, rather than Fedoraisms/Redhatisms as best supported scheme by systemd. For example, systems running systemd now generally store their hostname in /etc/hostname, something that used to be specific to Debian and now is used across distributions⁷.”

One of the main differences between traditional Upstart/SysVinit based Linux is that systemd doesn’t have **run levels**. The command `init 3` was always start at the commandline, and `init 5` was GUI. Systemd introduces **targets** in their place. Targets are supposed to be more flexible in what they can load and how they are loaded as opposed to the values of the `/etc/inittab` ⁸.

Run Level	systemd target	Function
0	runlevel0.target, poweroff.target	Shuts down the system
1	runlevel1.target, rescue.target	singleuser rescue mode
2,3,4	runlevelX.target, multi-user.target	multi-user text-mode
5	runlevel5.target, graphical.target	multi-user GUI mode
6	runlevel6.target, reboot.target	reboots
	emergency.target	Emergency shell

You can use the command `systemctl` to list all running services and to issue them commands. If you wanted to change to a graphical mode directly from your GUI mode you would issue `systemctl isolate graphical.target` effectively changing run levels or targets. You can change the default target by issuing this command: `systemctl set-default <name of target>.target` `systemctl get-default` will print out your current default target.

Exercise: Run the `systemctl get-default` command, what is the output?

10.7 Processes and Services

Whenever you start a program in Linux, whether that is a service, or something as simple as run a command in the terminal or open a new web-browser tab, that creates a system process. Each process gets an ID so that it can be accessed or referenced and is assigned memory space and CPU affinity (priority). In addition to a process—which can be short lived or long lived, there are services—which can be helper items such as the login and authentication service or something focused such as an apache2 web-server. Each process in turn has a PPID—Parent Process ID, which tells you which other process launched that process. In traditional SysVinit `/sbin/init` launches each additional service. In systemd, instead of having PIDs and PIDs, you have the concept of cgroups and processes grouped together.

10.7.1 Working With Services in systemctl

Example Usage: On a systemd based system, service control is done with `sudo systemctl <command> <name>.service`. In the case of Apache2 webserver the command to restart it would look like this: `sudo systemctl restart nginx.service` The `.service` can be left off and the system will assume the extension.

⁶<https://en.wikipedia.org/wiki/Systemd>

⁷<http://0pointer.de/blog/projects/the-biggest-myths>

⁸<http://fedoraproject.org/wiki/Systemd>

Example Usage: The `systemctl` command has additional abilities. It absorbed the `chkconfig` command, which was/is used to set services to autostart at boot time. In Fedora installed services do not automatically start at boot time, you must explicitly add them. You can check the status of the `httpd` service by issuing: `sudo systemctl is-enabled nginx.service`. Issue that command and what does it report?

Example Usage: To disable a service at boot type: `sudo systemctl disable nginx.service` and `enable` does the opposite. The `status` option will tell you what is the current status, start or stopped.

Systemd has the capability for targets to show what they “Require” to run, and what they “Want” to be running before they start. Systemd command `systemctl show` will show all details relating to a target or a service. Let us look at a service definition. Type this command, `cd /lib/systemd/system`, what do you see? Type this command to see the contents of the `httpd.service` file, `cat nginx.service` what do you see contained inside?

Compared to the make up of a SysVinit service, systemd has a simple design for each of its *service* files. They are located in `/lib/systemd/system`. Upon entering this directory you can launch an `ls` command and see the actual service files that you start/stop and enable/disable with `systemctl`.

```
# /lib/systemd/system/nginx.service file content
[Unit]
Description=A high performance web server and a reverse proxy server
Documentation=man:nginx(8)
After=network.target

[Service]
Type=forking
PIDFile=/run/nginx.pid
ExecStartPre=/usr/sbin/nginx -t -q -g 'daemon on; master_process on;'
ExecStart=/usr/sbin/nginx -g 'daemon on; master_process on;'
ExecReload=/usr/sbin/nginx -g 'daemon on; master_process on;' -s reload
ExecStop=-/sbin/start-stop-daemon --quiet --stop --retry QUIT/5 --pidfile /run/nginx.pid
TimeoutStopSec=5
KillMode=mixed

[Install]
WantedBy=multi-user.target
```

The `show` and `--property` options allow you to retrieve individual values from any service file without having to display the entire file.

- The command `sudo systemctl show --property "WantedBy" nginx.service`
 - Will show only the property “WantedBy” from the service file
- The command `sudo systemctl show --property "ExecStart" nginx.service`
 - Will show only the property “ExecStart” from the service file
- The command `sudo systemctl show --property "After" nginx.service`
 - Will show only the property “ExecStart” from the service file

10.7.2 Major systemd Components

Not just an init system replacement, systemd has replaced or merged the functionality of many other Linux services into systemd⁹. Beside its primary purpose of providing a replacement Linux init system, systemd suite provides additional functionality, including its following components:

journald `systemd-journald` is a daemon responsible for event logging, with append-only binary files serving as its logfiles. The system administrator may choose whether to log system events with `systemd-journald`, `syslog-ng` or `rsyslog`. The potential for corruption of the binary format has led to much heated debate.

logind `systemd-logind` is a daemon that manages user logins and seats in various ways. It is an integrated login manager that offers multiseat improvements and replaces `ConsoleKit`, which is no longer maintained. For X11 display managers the switch to `logind` requires a minimal amount of porting. It was integrated in systemd version 30.

⁹https://en.wikipedia.org/wiki/Systemd#Ancillary_components

networkd networkd is a daemon to handle the configuration of the network interfaces; in version 209, when it was first integrated, support was limited to statically assigned addresses and basic support for bridging configuration. In July 2014, systemd version 215 was released, adding new features such as a DHCP server for IPv4 hosts, and VXLAN support.

tmpfiles systemd-tmpfiles is a utility that takes care of creation and clean-up of temporary files and directories. It is normally run once at startup and then in specified intervals.

timedated systemd-timedated is a daemon that can be used to control time-related settings, such as the system time, system time zone, or selection between UTC and local time zone system clock. It is accessible through D-Bus. It was integrated in systemd version 30.

udev udev is a device manager for the Linux kernel, which handles the /dev directory and all user space actions when adding/removing devices, including firmware loading. In April 2012, the source tree for udev was merged into the systemd source tree.

libudev It is the standard library for utilizing udev, which allows third-party applications to query udev resources.

systemd-boot systemd-boot is a boot manager, formerly known as gummiboot. Kay Sievers merged it into systemd with rev 220. The systemd-boot tool is designed for hardware that is moving forward using the UEFI firmware instead of BIOS and is designed to handle Secure Boot as well. You can see a tutorial at <https://blobfolio.com/2018/06/replace-grub2-with-systemd-boot-on-ubuntu-18-04/> here on how to enable it.

systemd-homed systemd-homed.service manages home directories of regular (“human”) users. Each directory it manages encapsulates both the data store and the user record of the user so that it comprehensively describes the user account, and is thus naturally portable between systems without any further, external metadata¹⁰. This is a brand new paradigm and just released 10/26/20 – yet to be implemented in any Linux distros.

10.7.3 Systemd Service Types

Let’s look at the contents of a systemd unit file. Note it consists of basic INI style headers and compared to an rc file/script it is not a bash script. The full options are located at the systemd wiki <https://www.freedesktop.org/software/systemd/man/systemd.service.html>. The major units that systemd include are `.service`, `.mount`, `.timer`, `.target` and the entire list can be found <https://www.digitalocean.com/community/tutorials/understanding-systemd-units-and-unit-files>.

10.7.3.1 Systemd Service file

This is an example of the service file installed in Ubuntu 20.04 for Apache2 Web Server located in `/lib/systemd/system/apache2.service`. All system services are installed in this location. The parallel location in `/etc/systemd/system/` will append and override system defined settings.

```
#!/lib/systemd/system/apache2.service
[Unit]
Description=The Apache HTTP Server
After=network.target remote-fs.target nss-lookup.target

[Service]
Type=forking
Environment=APACHE_STARTED_BY_SYSTEMD=true
ExecStart=/usr/sbin/apachectl start
ExecStop=/usr/sbin/apachectl stop
ExecReload=/usr/sbin/apachectl graceful
PrivateTmp=true
Restart=on-abort

[Install]
WantedBy=multi-user.target
```

You will note the 3 headers listed in the file:

¹⁰https://systemd.io/HOME_DIRECTORY/

- Unit
- Service
- Install

Each has its own specific values. The name of the service file is important as well. This file name become the service name. The file `/lib/systemd/system/apache2.service` is responsible for the apache2 service. Under Unit, you have a description field which is a comment for the user. The next value is: `After=network.target`, this is a powerful addition to systemd that SysVInit did not have. This allows you to give the service a conditional start. In this case, only start the webserver after the `network.target` service starts, which makes sense.

In the Service tag, these are the commands to start and stop various shell scripts. When you use the `start | stop | reload | status` commands, these are the files or commands that are executed. The Install tag, is the final tag and tells systemd on which run level to start this service. Make note that the application uses absolute paths to all of the executables and binaries, this is do to the service run when parts of the operating system are still loading. FreeBSD still uses `rc` files which are shell scripts for starting services. You can find them listed in `/etc/rc.d/`. Take a look at `/etc/rc.d/syslogd` and you will see it is a 74 line shell script, compared to the 12 line systemd unit file.

10.7.3.2 Create a Service File for a User Created Script

This is a sample of a `.service` file created for a Python user script, called `write-journal.service`

```
[Unit]
Description=Script that writes a Hello message to the journal
After=network.target
```

```
[Service]
Type=simple
PIDFile=/run/bb.pid
User=controller
ExecStart=/usr/bin/python3 /usr/local/bin/write-journal.py
PrivateTmp=true
```

```
[Install]
WantedBy=multi-user.target
```

```
# .service file for the Rsyslog service:
# /lib/systemd/system/rsyslog.service
```

```
[Unit]
Description=System Logging Service
Requires=syslog.socket
Documentation=man:rsyslogd(8)
Documentation=http://www.rsyslog.com/doc/
```

```
[Service]
Type=notify
ExecStart=/usr/sbin/rsyslogd -n
StandardOutput=null
Restart=on-failure
```

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

10.7.3.3 Creating a `.timer` script

Systemd is efforting to replace cron jobs with a centralized systemd format called timers. You can see all the timers set on your system by default by using the `systemctl list-timers` command. The `.timer` files are named the same as the `.service` file – this correlates them. Timer files are stored along with service files in the `/lib/systemd/system` directory. Here is a sample `.timer`:

```

# This is the /lib/systemd/system/apt-daily.timer
# that runs /lib/systemd/system/apt-daily.service
# This does a daily update for packages via apt-get
[Unit]
Description=Daily apt download activities

[Timer]
OnCalendar=*-*-* 6,18:00
RandomizedDelaySec=12h
Persistent=true

[Install]
WantedBy=timers.target

```

The main variable here is the `OnCalendar` value, where you can define a time when the script takes place and or the frequency. It can use various types of time definitions¹¹. Here is a small list defining the various ways you can state time, even extending the traditional CronTab syntax and adding ranges (seperated by the two dots “.”).

- Wed 18:00:00
- Mon..Wed *-5-27
- 2020-05-27
- *:0/2
- 15/2
- --* 4:00:00
- hourly
- daily
- weekly
- monthly

10.7.3.4 Logging and Service Files

You will need to install some pre-reqs for this example.

```

# Ubuntu 20.04 - https://pypi.org/project/cysystemd/
sudo apt-get install build-essential \
    libsystemd-dev \
    python3-pip \
    python3-dev
python3 -m pip install cysystemd

# Fedora/CentOS - https://pypi.org/project/cysystemd/
sudo yum install gcc systemd-devel python3 python3-devel python3-pip
python3 -m pip install cysystemd

```

Create a python script called `write-journal.py` and include this code after installing the pre-reqs.

```

from cysystemd import journal

journal.write("Hello Lennart")

# Or send structured data
journal.send(
    message="Hello Lennart",
    priority=journal.Priority.INFO,
    some_field='some value',
)

```

Give the above script execute permission, execute it by typing `python3 write-journal.py`, and then execute the command: `sudo journalctl -xe`, what do you see?

¹¹<https://linuxconfig.org/how-to-schedule-tasks-with-systemd-timers-in-linux>

10.7.3.5 hostnamectl and timedatectl

One of the 69+ components of systemd is hostnamectl which is designed to give you an easy interface into controlling the information relating to your systems hostname. Running the command `man hostnamectl` shows you what can be done here [hostnamectl](#). The command `sudo hostnamectl set-hostname newhostnamehere` will change the displayed hostname of your system upon next session login.

Exercise: Use the hostnamectl command to change your systems hostname to itmo-556 (or your class name). Now close your shell and reopen it—what do you see?

The `timedatectl` is used for setting time zone and to activate ntp, [network time protocol](#), synchronization. This part of systemd supersedes previous commands that ran to handle the clock. `timedatectl`. The command `sudo timedatectl status` shows the current timedatectl configuration. You can also enable NTP and change the timezone via `timedatectl`, no need to use external NTP services anymore.

Exercise: Using the man command for `timedatectl` can you enable ntp synchronization? Can you change the timezone to UTC using the information at this URL: <https://www.freedesktop.org/software/systemd/man/timedatectl>?

10.7.3.6 systemd-analyze

Systemd was designed to bring modern OS principles to desktop and server Linux. That includes a tool called `systemd-analyze` which breaks down the time it took for all services, modules, and parts of the kernel to finish loading. To further debug these numbers use, `systemd-analyze blame`. This will print out individually which services/targets/units/mounts are taking the most time to load and allow you to investigate or disable those elements. You can even use the builtin *plotting* feature of `systemd-analyze`, by typing, `systemd-analyze plot > plot.svg` and then typing `eog plot.svg` to create a visual time based graph of your plot. There are additional commands under `systemd-analyze`, `critical-chain` will print specific load time for dependent services of the service you provide, `systemd-analyze critical-chain httpd.service`. These tools and options available in the man page, are used to determine system boot-up performance statistics.

10.7.4 Killing Processes with systemd

Systemd on the other hand has a mechanism for dealing with services directly, `systemctl kill -s SIGTERM httpd.service` will kill the Apache2 webserver service. You can issue the kill commands above within systemd individual services which are wrapped in control groups.

The systemd service using the `systemd-cgls` command and the `systemctl kill` command have effectively replaced the traditional use of `ps` by instead grouping processes into **cggroups**. Cgroups or Control Groups, allow for child processes to be grouped together, and concpets such as resource quotas or isolation can now be introduced. This is a more effective manner in killing all the child processes of a parent process. Cgroups are a Linux specific concept, which ultimately lead to the idea of OS containers and Docker—these are things that BSD and Unix don't have and can't run at the moment—which if OS containers might be the way of the future computing—doesn't bode well for BSD/Unix.

10.7.4.1 cggroups

Systemd uses cgroups as a way to hierarchically group and label processes, and (B) a way to then apply resource limits to these groups. cgroups allow you to *police* the resource usage of processes and related subprocesses—which gives finer grained control over other tools.

By typing the command, `systemd-cgls` you can see a ordered hierarchy of which processes are part of which cgroup. You don't have to search for process IDs anymore, you simply kill the entire cgroup.

Example Usage: To terminate the Apache2 web-server service, (assuming it has been enabled and started) first let's see the processes in its cgroup by typing `systemd-cgls`. You can filter just the apache2 process and sub-processes with the command: `sudo systemd-cgls -u apache2.service`. You can issue a kill command in the same way you can kill traditional processes by typing, `systemctl kill httpd.service`. You can also issue a kill level command through the `-s` flag, `systemctl kill -s SIGHUP httpd.service` will issue a `kill -1` command to all the members of the `httpd.service` cgroup.

10.8 Filesystems /proc

“/proc is very special in that it is also a virtual filesystem. It’s sometimes referred to as a process information pseudo-file system. It doesn’t contain ‘real’ files but runtime system information (e.g. system memory, devices mounted, hardware configuration, etc). For this reason it can be regarded as a control and information centre for the kernel. In fact, quite a lot of system utilities are simply calls to files in this directory. ¹²”

The /proc virtual filesystem provides you a file based interface to the processes that are running on your system. When you type `ls /proc` what do you see? You see a series of numerical directories. These numbers correspond to process IDs. Inside of each directory there are a series of files that represent the state of the process at the moment of introspection. This can be handy in debugging an application or fine tuning a system in regards to memory usage. Try to launch a Firefox or any other browser window. Use the `ps -C` command from above to find its process ID. Then find that process directory in /proc. What do you see? Some of the highlights are /proc/PID/cmdline, which will tell you what command line options were used in launching that particular process, /proc/PID/status links to the process status in human readable form, and /proc/PID/mem describes the memory held by this process. The command `procinfo` will give you summary of all system and resource states, the package may need to be installed. For an exhaustive list of all the contents and meanings you can find a chart at the Linux Documentation Project, <http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html>.

In addition /proc has convenient information about the state of your system. To display information about your processor you would type, `cat /proc/cpuinfo` this prints out the processor family, the feature flags, and the number of processors (physical and logical). The same can be done with memory by typing, `cat /proc/meminfo`.

The Linux kernel also has a concept of loadable kernel modules. These are pieces of code that can be added into or removed from the kernel—statically at boot, or dynamically as needed, so as to extend the capabilities of your kernel, without forcing unneeded code. For example—needing drivers for a floppy disk would be a waste to include that in the kernel when it could be added and removed via a loadable module if you happened to need it for testing. To list all the module currently loaded you would type, `lsmod`. This command is actually just formatting the content of /proc/modules. There are other shortcut commands as well to inspect system devices.

<code>lsmod</code>	Lists all currently loaded kernel modules
<code>lspci</code>	Lists all the currently detected PCI devices
<code>lsusb</code>	Lists all the currently detected USB connections
<code>lsblk</code>	Lists all block devices attached to the system (useful for hard-drives)
<code>lshw</code>	Lists detailed information on the hardware configuration of the machine

10.8.1 Loading Modules

You can list, load, and remove kernel modules from a running kernel. This is desirable because it allows you to change functionality on a permanent basis or temporary basis without having to recompile the core Linux kernel each time you make a change - hence loadable kernel modules. One of the best instances is **KVM** KVM stands for *Kernel-based Virtual Machine* and though present in the Linux Kernel the module is not loaded until you install the KVM software libraries that call for that module to be loaded on boot. If you had the KVM/Qemu virtualization applications installed (`sudo dnf install qemu-kvm libvirt`) then you would type `lsmod | grep kvm*` to see the modules loaded or type `sudo apt-get install zfsutils-linux` to load the ZFS kernel modules for Ubuntu.

Example Usage: You will notice for instance if you type `lsmod | grep vbox*` you will see VirtualBox kernel modules loaded - you wouldn’t see these if you were on a natively installed Linux system.

Example Usage: The `modprobe` command is a more intelligent way to add kernel modules than `insmod`. The command `lsmod` will list activated kernel modules and `rmmmod` will unload a kernel module.

You can also use the `systemctl` command with filter options to find modules that have loaded or failed to load.

Example Usage: You will notice your can filter the `systemctl` command: `sudo systemctl --failed`, the normal behavior is to show all running and failed.

¹²<http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html>

Example Usage: On Ubuntu, if you install the package, `zfsutils-linux` this will add a kernel module for the ZFS file system. Run the `lsmod` command before and after you install the ZFS package. What is the difference?

```
lvm2-lvmetad.service    loaded active running  LVM2 metadata dae
lvm2-monitor.service   loaded active exited  Monitoring of LVM
lxcfs.service          loaded active running  FUSE filesystem f
lxd-containers.service loaded active exited  LXD - container s
mdadm.service          loaded active running  LSB: MD monitorin
mysql.service          loaded active running  LSB: Start and st
● networking.service   loaded failed failed    Raise network int
ondemand.service       loaded active exited  LSB: Set the CPU
open-iscsi.service     loaded active exited  Login to default
```

Figure 10.6: `sudo systemctl -failed`

10.9 Single User Mode

If you have a system with an issue—or damage that needs to be repaired. Perhaps you need to reset the root password? You can drop your system into what was once known as single-user mode or `runlevel1.target` by issuing a command: `sudo systemctl isolate runlevel1.target` this command should be used sparingly because what it does is drop you to a commandline prompt with a single user logged in (root) with no password. This can be used to change or modify lost system passwords, or even reset database passwords or other troubleshooting issues like filesystem checks, which we will talk more about in the next chapter.

10.10 Chapter Conclusions and Review

Through this chapter we learned about init systems, the traditional SysVinit and the new systemd init commands. We learned about how to manage processes in both systems and the basics of how processes are handled. You learned about the `systemctl` command for managing processes. You learned about the `ps` command for managing processes under SysVinit. Finally we learned about the `/proc` virtual filesystem and how it presents process information in a file format dynamically on boot and during a system's use.

10.10.1 Review Questions

- 1) What is the name of the firmware that since 2015 has replaced BIOS on essentially all computers?
 - a) BIOS
 - b) POST
 - c) GRUB
 - d) UEFI
- 2) What is the name of the GNU software that is the first software program that runs when a computer with Linux installed starts?
 - a) BIOS
 - b) LILO
 - c) GRUB
 - d) GLOADER
- 3) In what Linux directory are the kernel and `initrd` image stored?
 - a) `/root`
 - b) `/root/kernel`
 - c) `/boot`
 - d) `/boot/vmlinux`
- 4) What is the name of the pre-kernel gzip file located in `/boot` that helps the kernel load?

- a) vmlinuz
 - b) initrd
 - c) initram
 - d) init
- 5) Where is the file location where the GNU Grub configuration is stored (on Ubuntu) that a user would edit?
- a) /boot/grub/grub.cfg
 - b) /etc/default/grub
 - c) /etc/grub/grub.cfg
 - d) /boot/kernel/conf
- 6) In the /etc/default/grub file, which of these options below would you edit to display the *splash* screen on boot so kernel messages are displayed?
- a) GRUB_CMDLINE_LINUX_DEFAULT
 - b) GRUB_BACKGROUND
 - c) GRUB_GFXMODE
 - d) GRUB_TIMEOUT
- 7) What is the command to make changes to /etc/default/grub permanent?
- a) No special command just edit and save /etc/default/grub
 - b) sudo apt-get update
 - c) sudo update-grub
 - d) sudo updatedb
- 8) Under SysVinit - what is the ancestor process that launches first and every other process is started by it?
- a) root
 - b)/sbin
 - c) init
 - d) systemd
- 9) Under SysVinit - what runlevel is considered multi-user command-line only?
- a) 1
 - b) m
 - c) 3
 - d) 5
- 10) Which Operating System is still using the Upstart init system?
- a) Ubuntu
 - b) MX Linux
 - c) Fedora
 - d) ChromeOS
- 11) What is the name of the init system that has replaced SysVinit in every single major Linux distribution (not including Devuan and Gentoo Linux)?
- a) systemX
 - b) systemd
 - c) systemV
 - d) initrd
- 12) What is the name of the command you use in systemd to inspect, start, stop, and modify process states?
- a) systemd
 - b) systemd-init
 - c) service
 - d) systemctl
- 13) How would you start the `nginx.service` on an Ubuntu system using systemd?
- a) `sudo system start nginx`

- b) `sudo service start nginx`
 - c) `sudo systemctl nginx start`
 - d) `sudo systemctl start nginx`
- 14) What would be the command to disable (make the service not start at boot time) the `httpd` service on Fedora using `systemd`?
- a) `sudo service httpd stop`
 - b) `sudo systemctl disable apache2.service`
 - c) `sudo systemctl stop httpd.service`
 - d) `sudo systemctl disable httpd.service`
- 15) What is the Linux command to inspect processes (not part of `systemd`)?
- a) `p`
 - b) `ps`
 - c) `proc`
 - d) `meminfo`
- 16) `SysVinit` used the concept of PIDs and PPIDs—what did `systemd` replace these with?
- a) `proc-groups`
 - b) `sys-groups`
 - c) `cgroups`
 - d) `xgroups`
- 17) According to Lennart, how is `systemd` defined?
- 18) The `/proc` filesystem provides you what? (choose all that apply)
- a) Provides you a file based interface to the processes that are running on your system
 - b) It can be regarded as a control and information center for the kernel
 - c) Shows up to the second process usage—updated in real time
 - d) Is a replacement for the `top` command
- 19) What command can be used to list all the `pci` devices attached to your system?
- a) `ls -pci`
 - b) `ls -p`
 - c) `lsusb`
 - d) `lspci`
- 20) What is the runlevel target that has a single user only as root, using no password: commonly called single-user mode?
- a) `runlevel3.target`
 - b) `runlevel5.target`
 - c) `runlevel0.target`
 - d) `runlevel1.target`

10.10.2 Podcast Questions

View the presentation by FreeBSD developer Benno Rice from BSDCan 2018 at <https://www.youtube.com/watch?v=6AeWulfZ7bY> and answer the following questions:

- 1) ~1:00 Who is Benno Rice?
- 2) ~1:31 What is Contempt Culture?
- 3) ~3:21 What is `inits` job?
- 4) ~6:11 What led to the concept of a service?
- 5) ~8:35 What does the traditional `rc` system not do?
- 6) ~9:27 What OS had a strong initial concept of services from the beginning?
- 7) ~10:00 On MacOS what did `launchd` replace?
- 8) ~11:53 In 2010 What was Lennart Poettering looking at?
- 9) ~13:48 What other service did Lennart say he was heavily borrowing from?

- 10) ~14:01 What does Lennart say that systemd is about?
- 11) ~14:43 What is the layer in-between the kernel and the userspace created by systemd?
- 12) ~17:11 Does systemd violate the UNIX philosophy?
- 13) ~20:33 What does Benno think is incredible about what Lennart accomplished?
- 14) ~25:26 Why is using systemd as a recruiting tool for BSD (which doesn't have it) a bad idea?
- 15) ~28:20 What are a few features that BSD could gain from systemd?
- 16) ~28:20 Why can't BSD run containers?

10.10.3 Lab 10

10.10.3.1 Lab 10 Objectives

- Learn how to modify GRUB settings
- Use `systemctl` to start, stop, and examine processes in systemd
- Use `systemd-analyze` to understand what services are loading during system boot
- Learn how to change systemd targets
- List the kernel modules currently loaded on your Linux system

10.10.3.2 Lab 10 Outcomes

At the conclusion of this lab, you will be able to manage, edit, and list system processes in systemd—helping you to master the concepts of systemd. After each item take a screenshot and place it below the question to demonstrate the answer (unless specified otherwise). Edit your screenshots to show just the relevant information.

- 1) Take a screenshot of the Advanced GRUB boot settings in an Ubuntu virtual Machine (you can access this menu by starting a virtual machine from a cold start, click your mouse into the virtualbox window immediately after the VM starts and hold the **shift** key down until you see the GRUB menu). Select the Advanced option and take the screenshot of the kernels and the recovery options.
- 2) Change the default GRUB settings on your Ubuntu virtual machine uncommenting the entry `GRUB_DISABLE_RECOVERY="true"`. Save the changes the GRUB configuration file, reboot the virtual machine, repeating the process in the first question, and now take a screenshot of the same menu that is missing the recovery options.
- 3) Use the `systemd-analyze` tool to print out the most recent boot times for your Fedora virtual machine.
- 4) Use the `systemd-analyze` tool to print out the most recent boot times for your Ubuntu virtual machine.
- 5) Install MariaDB server on Fedora, `sudo dnf install mariadb-server`. Use the command `sudo systemctl status <servicename>` after MariaDB is installed to take a screenshot of the display of its current status. Enable the service via `systemctl`, and then start the service. Finally, reboot your system. Upon reboot, take a screenshot of the output of the `systemctl status` command for `mariadb.service`.
- 6) With MariaDB enabled on Fedora, use the `systemd-analyze` tool to print out the most recent boot time for your system again and compare this to the first boot time screenshot to see if adding this service increased the boot time.
- 7) Use `systemctl` to enable and start the `nginx.service` (Fedora), open a webbrowser and navigate to `http://127.0.0.1`. Then issue the `systemctl` command to stop the `nginx.service`. Navigate again to `http://127.0.0.1` and take a screenshot.
- 8) Change the `systemd` target to the `systemd` commandline-only level, display the `systemd` default target level. Then change the run level back to the GUI target (or `runlevel5`)
- 9) Using `systemctl` and the `--show` option, display the “After” and “WantedBy” properties of the `sshd.service`.
- 10) Type one of the two commands mentioned in the chapter to display info about your CPU hardware (a single screenshot will do incase the output scrolls past one screen).
- 11) Using `systemctl status` command, find and display the status and cgroup information for the `nginx` webserver and take a screenshot. Then issue a `kill -s SIGHUP` command to that service and take a screenshot of the results.
- 12) Using `systemd-cgls -u nginx.service` command and the SysVinit command, `ps -ef | grep nginx`, take a single screenshot of the combined output of the `httpd.service` process IDs.

- 13) show the output of the `timedatectl status` command before and after the timezone change. Use the `timedatectl` command to change the clock of your system to UTC.
- 14) Use the `hostnamectl` command to:
 - a) set-hostname to itmo-556-xyz (xyz is your initials)
 - b) set-location to: dlr1u22
 - c) set-chassis to: vm
 - d) set-deployment to: development
 - e) display the changes by issuing the `hostnamectl` command
- 15) What would be the command to change the systemd target runlevel to recovery mode? Execute this command and take a screenshot of the result – then change it back to the regular GUI interface.
- 16) Review the content of the `mariadb.service` file. List the contents of the `[Install]` header that must load before and after the `mariadb` service starts (assume that `mariadb-server` package is installed)
- 17) Find the sample file located in: `files > Chapter-10 > python > iloop.py`. Copy this file to `/usr/local/bin` and then execute the file by typing: `iloop.py`, type `Ctrl+C` to exit: take a screenshot of the output
- 18) Copy the sample template file located in: `files > Chapter-10 > python > iloop.service` to your `/lib/systemd/system` directory. Edit the `iloop.service` file adding a description, type `simple`, `private` `tmp true`, `wantedby multi-user.target`, and `ExecStart` the absolute path to the `python3` binary and absolute path to `iloop.py`. Take a screenshot of the output of the `systemctl start` command and then the `status` command.
- 19) Using the `systemd-cgls` command find the `cgroup` ID for the `iloop.service` and take a screenshot of that entry. After taking the screenshot execute the `systemctl stop` command for the `iloop.service`.
- 20) Take a screenshot of the output of the command: `systemctl list-timers`, make sure the terminal is expanded to include the `UNIT` column in the screenshot.
- 21) Copy the sample template file located in: `files > Chapter-10 > python > iloop.timer` to your `/lib/systemd/system` directory. Edit the `.timer` file configuring the `onCalendar` value for execution 3 minutes from the current time. This is so you can see the results of the scheduled task execute. Take a screenshot of the output of the command: `systemctl list-timers` showing the `iloop` timer as active, make sure the terminal is expanded to include the `UNIT` column in the screenshot.
- 22) Find the sample file located in: `files > Chapter-10 > python > write-journal.py` and copy it to `/usr/local/bin`. Execute the command `write-journal.py`. Use the `journalctl -xe` command and show the output message in the `journald` logs.

10.10.3.3 Footnotes

Chapter 11

Creating, Managing, and Mounting Filesystems

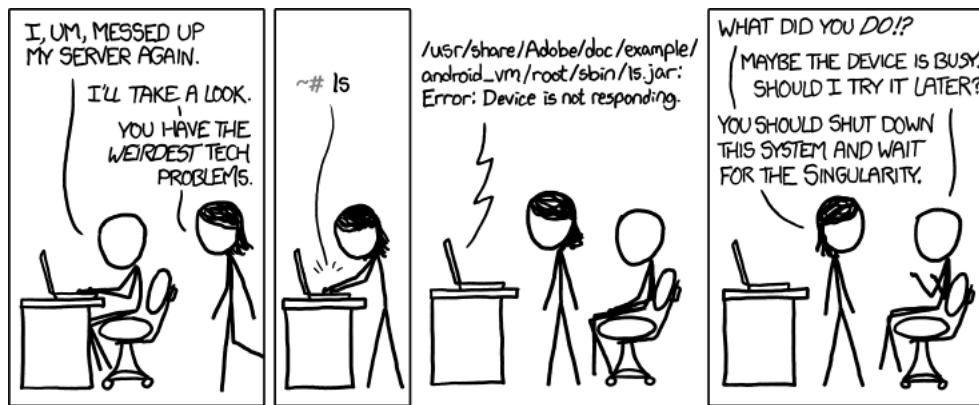


Figure 11.1: *Easy as cake...*

11.1 Objectives

- Compare and contrast different Linux filesystems
- Understand how to create and attach virtual disks
- Understand how the `fdisk` command is used to list, modify, and create filesystem partitions
- Understand Linux tools relating to filesystems, disk utilization, and mounting
- Understand how Logical Volume Management, extents, and disk based partitions differ
- Understand compression and archiving tools and their use on the command line

11.2 Outcomes

At the conclusion of this chapter you will be able to add additional virtual disks to any Linux version installed in VirtualBox. You will be able to format devices and create filesystems on newly formatted devices. You will understand how to partition a device and install filesystems. This chapter will give you experience with the tools needed to perform these actions. We will study the LVM – Linux Volume Manager and its new concept for dealing with disks. Finally we will learn the concept of mounting and unmounting of disks.

11.3 Storage Types

In looking at the prices of disk based storage from the Fall of 2015, a single terabyte Western Digital Blue hard drive was selling for ~\$50. In the time since, the price of storage has only decreased. Storage is cheap and with that in mind, adding *storage* or *capacity* to a system is trivial. In fact the types of storage available since 2015 have changed drastically.

11.3.1 Mechanical Hard Drives HDDs

The original, cheapest, and densest storage type in dollars per megabit is still a mechanical hard drive. Often referred to as an HDD¹ or magnetic drive.



Figure 11.2: *Hard Drive*

These are made up of spinning platters where bits are stored via magnetic charge. These systems have down sides in that parts of the surface wear out over time as well as they have mechanical parts (servos) that can fail over time. They require constant amounts of power and if you scale this over large data centers, these costs can quickly add up. In addition as the size of storage density has increased, the need to develop new storage mediums has arisen. What was once a single magnetic platter, became three parallel platters, then became five spinning platters, which then became glass platters, which then became vertical electron based storage (somehow they got electrons to stand up instead of lay flat...), then moving to the latest [helium filled 10 TB disks](#).

As disks became bigger, new patterns for reading and writing data were created: SATA 1, 2, and 3.x bus technology was introduced for transmitting data faster from the disk to the CPU. On-disk cache memory of 16-64mb was added to the disk for caching frequently accessed data. On top of that we have [NCQ](#). Native Command Queueing was introduced which looks at disk requests on the drive and reorders them to reduce round trips that a disk needs to make. All of this is happening at 5400-7200 rpms, revolutions per minute, at a fly height of 8 Pico meters!

11.3.2 Solid State Drives SSDs

In early 2012, a new medium called a Solid State Drive, or SSD was released. These drives were different than mechanical disks because they relied on Flash Memory (SLC or MLC) and had disk based controllers to address this data. The immediate advantage was that electrons move at the speed of light so the access time of any single bit was identical, compared to a mechanical drive which had to rotate into position to read the correct bits. This increased speeds dramatically and while the original SSDs storage was low and comparatively today the dollar per megabit ratio is not as good as a HDD, the read based access time was orders of magnitude faster. Version of SATA beyond 3.2 introduced a process called [SATAe](#), which focuses on using the PCI Express bus to increase performance, rather than changing the SATA standard.

SATA Version	Throughput
SATA revision 3.4	features added
SATA revision 3.3	features added
SATA revision 3.2	16 Gbit/s 1.97 GB/s
SATA revision 3.0	6 Gbit/s 600 MB/s
SATA revision 2.0	3 Gbit/s 300 MB/s
SATA revision 1.0	1.5 Gbit/s 150 MB/s

¹CC BY-SA 3.0 By Evan-Amos from Wikimedia Commons



2

The original SSD disks were put out by RAM manufacturers and had RAM controller write issues. The good ones were built by Intel and Samsung. The other manufactures have caught up and you can get drives in 256, 512, and 1 TB sizes. Since there are no moving parts the battery or power usage is far reduced from an HDD. SSDs do have potential issues with wear leveling and block failure, but in the chips controlling these devices the manufacturers have built in protection against failing flash memory chips to spread out the disk writes to prolong their lives. These SSD drives use the standard SATA data transfer protocols allowing them to be drop in replacements and allowing the initial bandwidth limitation that HDDs suffered to be overcome.

As a price point marketing creation you might see SSHDs which are called Hybrid Drives. They contain 4 to 16 GB of flash disk and the rest of the drive is a mechanical 3 to 5 platter disk. This is supposed to give you the advantage of caching frequent data on the flash data, but the idea never caught on as the price of SSD and HDD both have dropped drastically, making this solution not necessary.

11.3.3 NVMe

The latest incarnation of SSD disk based technology is [NVMe](#) which stands for Non-Volatile Memory Express. These are solid state drives, but instead of sending data over the SATA interface, they connect directly to the high-speed PCIe bus, just like your video card does. The advantage is in using the PCIe bus³ you gain the ability to transmit in 1 to 4x parallel transfer lanes as opposed to a serial fashion which SATA was designed for. NVMe comes in expansion card factor and in the smaller [M.2 form factor](#)⁴. The price is high as the technology is still relatively new, but the performance gain is worth the investment.

PCIe Version	Per Lane Throughput	1x	4x	8x	16x
PCIe 1.0	2 Gbit/s (250 MB/s)	2 Gbit/s	8 Gbit/s	16 Gbit/s	32 Gbit/s
PCIe 2.0	4 Gbit/s (500 MB/s)	4 Gbit/s	16 Gbit/s	32 Gbit/s	64 Gbit/s
PCIe 3.0	7 Gbit/s	7 Gbit/s	28 Gbit/s	56 Gbit/s	112 Gbit/s
PCIe 4.0	15 Gbit/s	15 Gbit/s	60 Gbit/s	120 Gbit/s	240 Gbit/s

The future of disk is something of a hybrid between ram and solid state/flash memory. Intel launched a trademarked platform called [Optane](#). The target is cloud based servers running OS Containers and Virtualized platforms. The idea is to increase the speed of disk to the point that is is close to or equal in speed to RAM (Non-volatile memory), thereby eliminating potential memory bottlenecks.

11.3.4 Virtual Hard Drives

When dealing with Virtual Machines, we can attach and detach storage very easily. With large deployments of VMware, and Cloud based services, in-place disk can be reformatted and used to attach virtual disks to a virtual machine - with the added ability to manipulate these hard drives as if they were simple files. This [interface was standardized](#) and opensourced as part of the [virtio](#) package.

11.3.5 Disk Management in VirtualBox

For this chapter we will assume that you are using VirtualBox 6.x, but these concepts apply to any virtual machine or Hypervisor. This also assumes you have free space on your computer. Since the point of this lab is to explore and not production usage, you may want to get an external USB hard drive and use that for this chapter so as not to fill up your hard drive.

²By photo: Qurren (talk)Taken with Canon IXY 10S (Digital IXUS 210) or CC BY-SA 3.0

³<https://www.lifewire.com/pci-express-pcie-2625962>

⁴CC BY-SA 4.0 By Anand Lal Shimpi, anandtech.com via Wikimedia Commons)

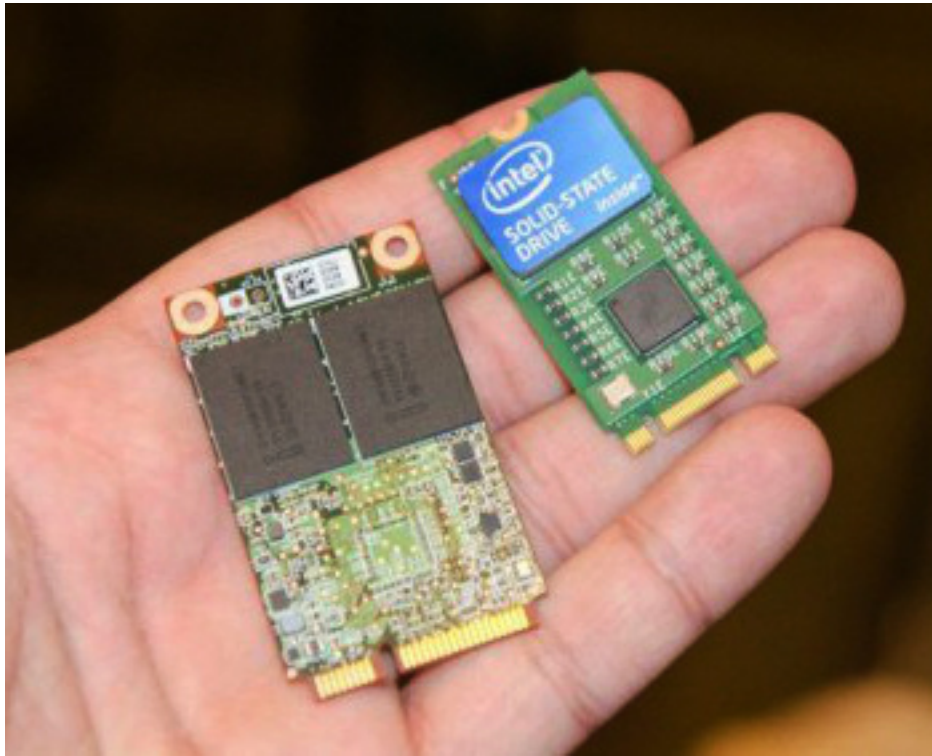


Figure 11.3: *mini-Sata and M.2*

With your Ubuntu or Fedora virtual machine powered down, let's add some new disks (virtually) to your Linux system. The first thing to do is locate the *SETTINGS* button on the VirtualBox main menu.

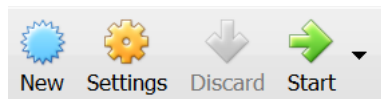


Figure 11.4: *VirtualBox settings panel*

The next menu to come up will show the *SETTINGS* options and the name of the virtual machine you are working on.

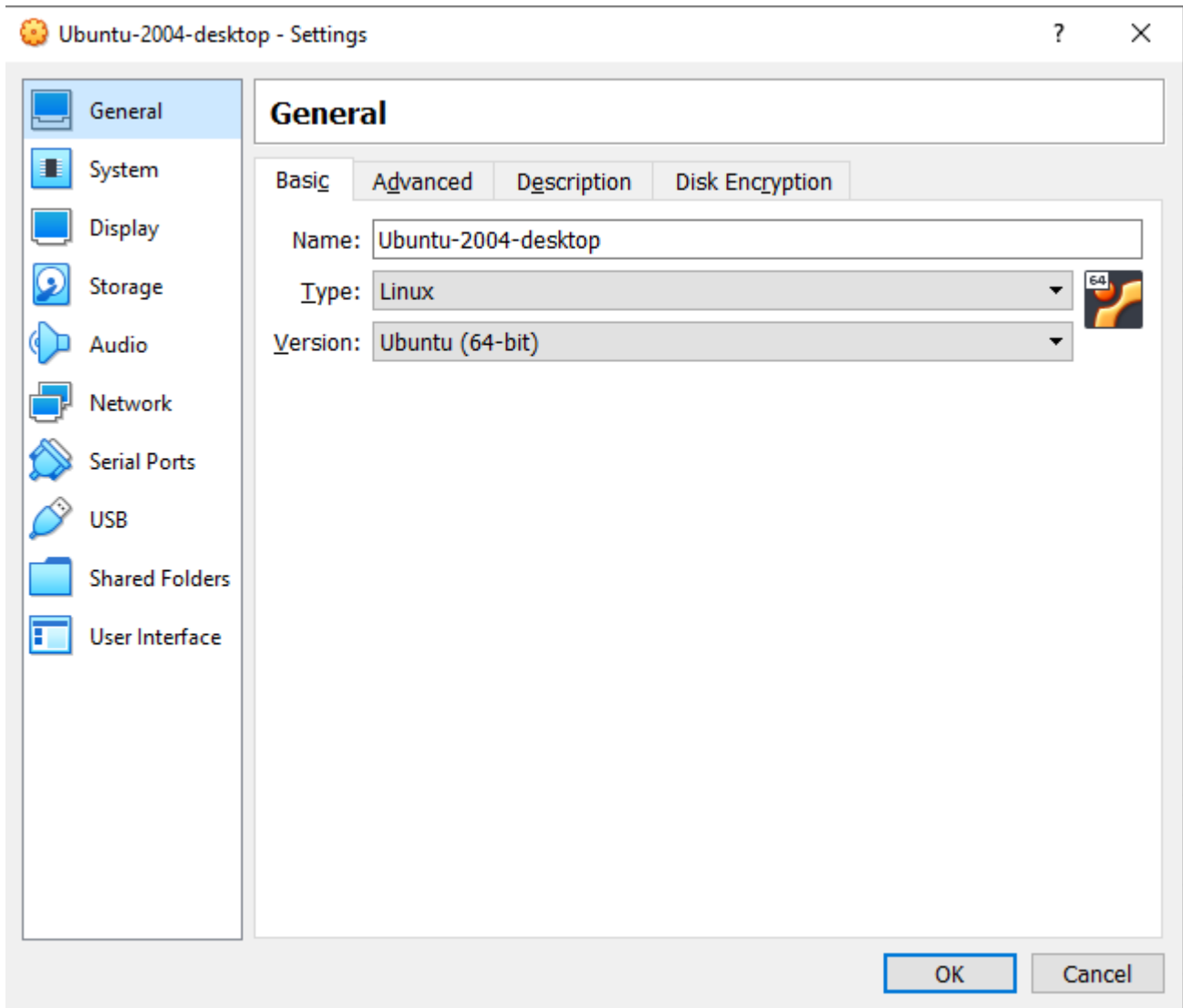


Figure 11.5: *VirtualBox settings menu*

Select the *STORAGE* option from the menu on the left—this is where you can attach, detach, and modify virtual disks in VirtualBox. In most cases these will be hard drives, but there is the ability to attach ISO images to a virtual cd-rom device as well. That option is in the top half where you see *Controller: IDE*. Under that you might see the term *EMPTY* or you might see a virtualbox-guest-additions.iso attached.

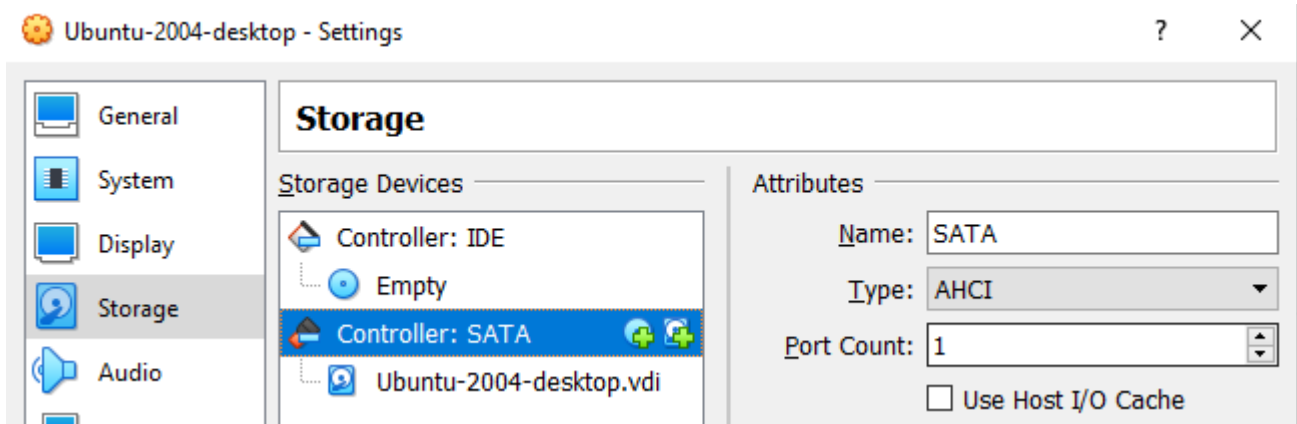


Figure 11.6: *Storage menu*

We will be working with attaching virtual hard drives so we are interested in the bottom portion of the menu which is identified by *Controller: SATA* which is your [Serial ATA](#) hard drive bus connection. As a refresher, Serial ATA is the name of the signaling protocol the operating system uses to retrieve data from a hard drive. Go ahead and highlight the SATA Controller entry.

In order to add a new hard drive to your virtual machine, click the blue HDD icon with a + sign at the bottom of the menu.



Figure 11.7: *Add Storage Icon*

Upon completion of that step a new menu will pop out of the HDD icon and give you an option to *Add Hard Disk*.

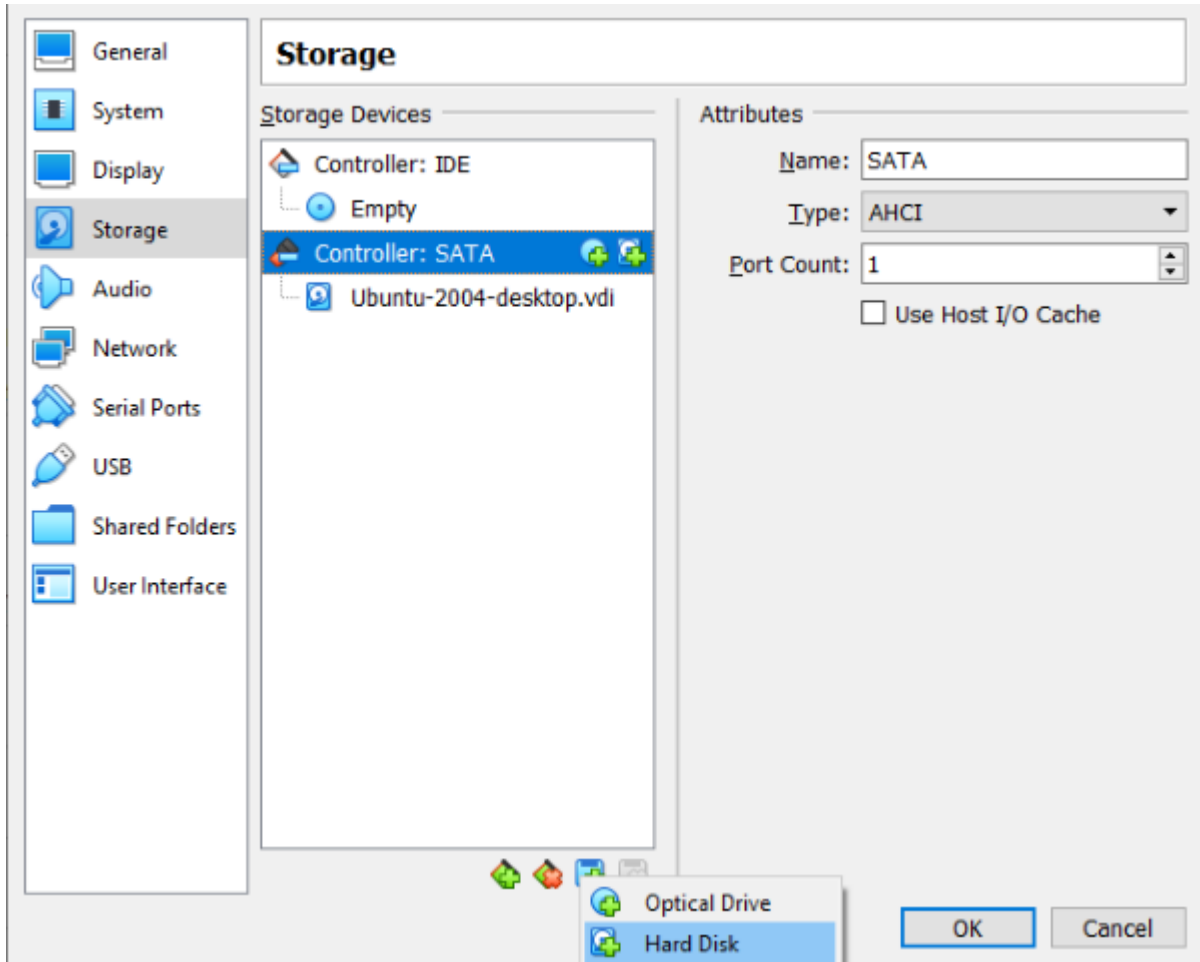


Figure 11.8: *Add Storage*

Once you have selected *Add Hard Disk* a familiar set of screens come up, these are the same screens you walked through in chapter 2, and the same set of screens you walk through when setting up a new virtual machine. You are presented first with an option to create a new disk or attach an existing one. Usually you want to create a new disk.

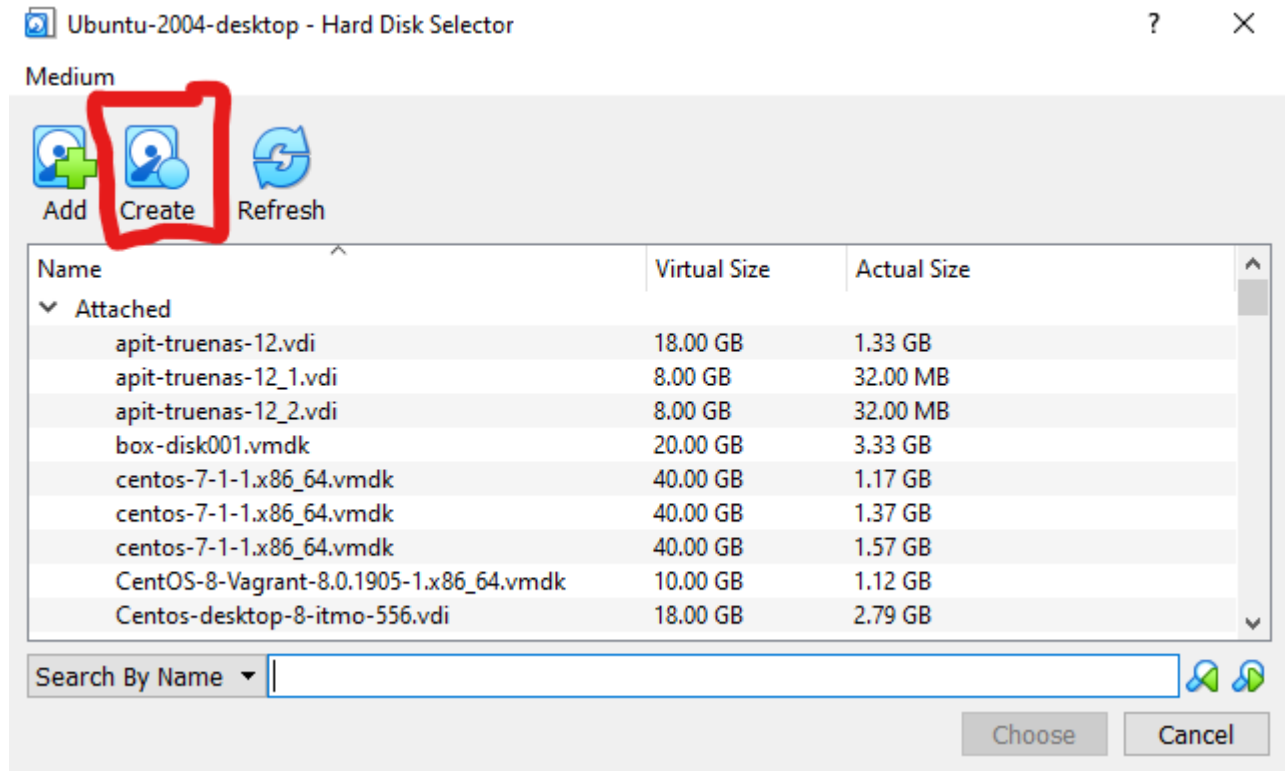


Figure 11.9: *Create New Disk*

Once that is selected you will be presented with the virtual disk type screen. Since we will be working with VirtualBox, the default setting of VDI (VirtualBox Disk Image) will be the best selection. But if you know this VM will be moving to another platform—you may want to choose accordingly.

Hard disk file type

Please choose the type of file that you would like to use for the new virtual hard disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.

- VDI (VirtualBox Disk Image)
- VHD (Virtual Hard Disk)
- VMDK (Virtual Machine Disk)

Figure 11.10: *VDI step-through*

The next page allows you to choose either a dynamic or static allocated hard drive. Dynamic is usually the best when you are working on a laptop or other development system, as you will be creating and destroying virtual machine rapidly, and static allocations of multiple gigabytes can become an issue after some time due to your disk filling up.

Storage on physical hard disk

Please choose whether the new virtual hard disk file should grow as it is used (dynamically allocated) or if it should be created at its maximum size (fixed size).

A **dynamically allocated** hard disk file will only use space on your physical hard disk as it fills up (up to a maximum **fixed size**), although it will not shrink again automatically when space on it is freed.

A **fixed size** hard disk file may take longer to create on some systems but is often faster to use.


- Dynamically allocated
- Fixed size

Figure 11.11: *Dynamic Filesystem*

This next screen allows you to change the location of where the virtual hard disk will be stored, as well as adjust the size of this new partition.

File location and size

Please type the name of the new virtual hard disk file into the box below or click on the folder icon to select a different folder to create the file in.

C:\Users\palad\VirtualBox VMs\Ubuntu-2004-desktop\Ubuntu-2004-desktop_1.vdi 

Select the size of the virtual hard disk in megabytes. This size is the limit on the amount of file data that a virtual machine will be able to store on the hard disk.



The final step is to choose to attach the virtual disk you just created to the virtual machine.

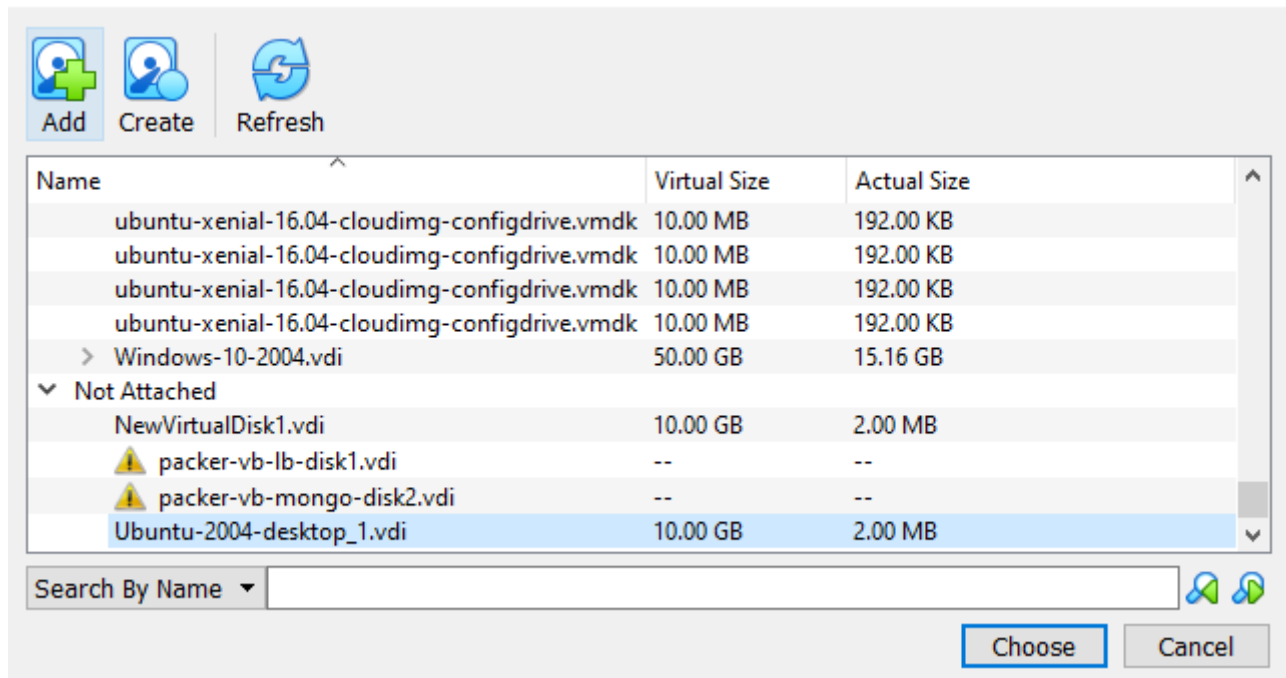


Figure 11.12: Choose new Virtual Disk to attach

11.4 Disk Partitioning and Formatting

Adding a virtual disk is only the first step, there are three more steps before we can use this disk. Though the first two steps have been largely merged into a single step over the last decade.

- 1) Partition the disk / create a filesystem
- 2) Mount the disk so that it can be used by your operating system.

According to the `fdisk` man page, `fdisk` is a dialog-driven program for the creation and manipulation of partition tables. The term **partition** in relation to a hard drive is an important concept. You can think of a brand new hard drive as a large plot of land, multiple acres of land. The land itself in that form is not very useful, just as a new hard drive added into your system is not very useful. Just as that land needs to be partitioned up into different uses and functions, a hard drive needs to know where its partitions are. Each disk can have multiple partitions. The `fdisk` method here is mentioned for historic reasons but is not used the installation of modern operating systems, but it is good to know where we came from.

Linux inherited a way to name each device and reference certain partitions attached to a system. Windows simply uses the letter C, D, E, and so forth. Linux and Unix use a device/partition nomenclature. You can see this currently by typing the `lsblk` command, which will print out currently all the block devices, their device name and their partitions in a nice tree based format.

```
controller@controller-VirtualBox:~$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda         8:0    0   18G  0 disk
├─sda1      8:1    0   16G  0 part /
├─sda2      8:2    0    1K  0 part
└─sda5      8:5    0    2G  0 part [SWAP]
sdb         8:16   0    4G  0 disk
sdc         8:32   0    4G  0 disk
sr0        11:0    1  1024M  0 rom
```

Figure 11.13: *lsblk* output from a virtual machine with 2 additional drives attached

Here you will note that the drives are references by the prefix **sdx** with the **x** being the alphabet letter in incremental order. Meaning that the first disk drive that your system detects is labeled **sda**, the next one would be **sdb**, and can you guess what the third and fourth system would be? In the image above you notice that **sda** has 3 partitions, **sda1**, **sda2**, and **sda5**. These three partitions were created at installation time by the default Linux installer. The first partition you can see has the character **/** in the far right column. That is where the **root** partition is mounted (meaning your entire filesystem). The second partition is where the **/boot** partition is mounted, and the final partition says **SWAP** in the far right meaning this is a Linux SWAP partition—used by the operating system for moving data in and out of RAM in chunks at a time or called *pages*.

You can create partitions on a new disk for a fresh OS installation or just create a single partition to contain data. The program mentioned above to create partitions is a program called **fdisk**. The **fdisk** command is considered an essential and standard Linux tool and is part of the [util-linux](#) package. The best command to get started with when dealing with new disks and creating partitions is `sudo fdisk -l`. This command will list the current existing disks and any partitions they may have. It will also report the undetermined state of any newly attached disks. See the image below for a sample output. If you are using Fedora 22/23 you will see a bit of a different output, you will see partitions labeled **LVM** which will be explained at the end of the chapter.

```

Disk /dev/sda: 19.3 GB, 19327352832 bytes
255 heads, 63 sectors/track, 2349 cylinders, total 37748736 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00035d66

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1  *          2048     33554431   16776192   83   Linux
/dev/sda2                33556478   37746687    2095105    5   Extended
/dev/sda5                33556480   37746687    2095104   82   Linux swap / Solaris

```

Figure 11.14: `sudo fdisk -l`

```

Disk /dev/sdb: 4294 MB, 4294967296 bytes
255 heads, 63 sectors/track, 522 cylinders, total 8388608 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

Disk /dev/sdb doesn't contain a valid partition table

```

Figure 11.15: `sudo fdisk -l`

The history of the Linux `fdisk` command goes way back. Stemming from the early 1990's hard drives at that time using the standard BIOS of the day were only allowed 4 **primary partitions** on the operating system. At those times, hard drives were small, and devices were expensive, and things we take for granted now, like optical drives, didn't really exist, so 4 primary partitions was thought to be more than anyone would ever need. A primary partition could be broken up into an **extended partition**. Then each **extended partition** could be further sub-divided into as many **logical partitions** that fit on the drive. At that time only one **primary partition** could be active (or bootable and seeable) at a time, all other primary partitions would be hidden from the currently active operating system. In this world `fdisk` was built, hence its concern with partitioning. There has been an improvement since 2000 called LVM, which is covered and thankfully used almost exclusively now by default.

To work/modify a device that has no existing partitions (say `sdb` in the image above). From the TLDP documentation regarding how to use `fdisk`:⁵ `fdisk` is started by typing (as root) `fdisk device` at the command prompt. Device might be something like `/dev/hda` or `/dev/sda` (see Section 2.1.1). The basic `fdisk` commands you need are:

- `p` print the partition table
- `n` create a new partition
- `d` delete a partition
- `q` quit without saving changes
- `w` write the new partition table and exit

To successfully create a partition on a new drive, let's select `sdb` in the example above. The command `sudo fdisk /dev/sdb` will enter into `fdisk` and operate on this device. Remember all *devices* are accessed through file handles in the `/dev` directory. Upon executing this command you are greeted with a status message reporting that the partition type cannot be detected or is not valid. The error message seems a bit dated because you notice that it mentions DOS, SUN, SGI, and OSF—all outdated or unused partition types. Similar to different languages or dialects, a partition also has to speak to an Operating system, and each operating system does it a bit different because of how particular filesystems are architected. Fortunately this is a simple choice for us as we only need a Linux and a Linux SWAP partition for our uses—the rest are just artifacts of the past.

⁵http://tldp.org/HOWTO/Partition/fdisk_partitioning.html


```
Device contains neither a valid DOS partition table, nor Sun, SGI or OSF disklabel
Building a new DOS disklabel with disk identifier 0x3ff33a95.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

Command (m for help): █
```

Figure 11.16: `sudo fdisk /dev/sdb`

Always type **m** for menu because the single letter commands are not intuitive.

```
Command action
 a  toggle a bootable flag
 b  edit bsd disklabel
 c  toggle the dos compatibility flag
 d  delete a partition
 l  list known partition types
 m  print this menu
 n  add a new partition
 o  create a new empty DOS partition table
 p  print the partition table
 q  quit without saving changes
 s  create a new empty Sun disklabel
 t  change a partition's system id
 u  change display/entry units
 v  verify the partition table
 w  write table to disk and exit
 x  extra functionality (experts only)
```

Figure 11.17: *m for menu*

If you type the letter **l** you will see the entire list of possible partitions, we are only interested in the value hex 82 and 83. The next command to type is **p** for printing out the current partition table—which will be blank.

```

Command (m for help): p

Disk /dev/sdb: 4294 MB, 4294967296 bytes
255 heads, 63 sectors/track, 522 cylinders, total 8388608 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x3ff33a95

   Device Boot      Start         End      Blocks   Id  System

```

Figure 11.18: *p for print*

The next step is to type the **n** command to create a new partition. You will be presented with two choices for your new partition. In this case you can select **primary partition**. In most cases in creating data drives you can select primary partitions without concern. If you find yourself creating many data drives or creating triple and quad bootable systems (multiple operating systems) then you will want to conserve those primary partitions and use **extended/logical** partitioning.

```

Partition type:
  p   primary (0 primary, 0 extended, 4 free)
  e   extended

```

Figure 11.19: *n is for new partition*

You are then presented with a series of options to choose a partition number, the beginning sector of your partition (usually best to choose the default) and the finishing sector (how big do you want your disk). You can specify in a known quantity of K=kilobytes, M=megabytes, G=Gigabytes and prefacing that value with a +. Selecting the default for the option of last sector will automatically fill up your disk with 1 single large partition.

```

Partition number (1-4, default 1): 1
First sector (2048-8388607, default 2048):
Using default value 2048
Last sector, +sectors or +size{K,M,G} (2048-8388607, default 8388607):

```

Figure 11.20: *New Partition Options*

Let's see if our partition was created successfully. You can type **m** to display the menu again or type the **p** directly to print out the current partition table. You will notice that it has been modified.

Device	Boot	Start	End	Blocks	Id	System
/dev/sdb1		2048	8388607	4193280	83	Linux

Figure 11.21: *Successful Partition Creation*

Everything looks good, but DON'T QUIT YET! If you type **q** now your changes will not be saved, and no partition information will be written. Now you need to type **w** to write the new partition data to the disk you are working on. The **w** command will write and quit out automatically for you. After writing this partition data, you will see if show up in the `sudo fdisk -l` command. After you see your new partition in `fdisk` of `lsblk` you are ready to move on to the next step of formatting a partition with a filesystem.

```
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

Figure 11.22: *Write the Partition table data to disk*

11.4.1 Drawbacks of disk partitions

Using the `fdisk` command does have its drawbacks. The tool was designed in the day when systems had 1 or 2 hard drives. Filesystems handled small files and the idea of large files partitions that spanned multiple disks was not possible due to software and processor limitations. But we see with the cost of disk alone, systems having 24 or more hard drives is not unheard of. Most filesystems were designed around the limitations of `fdisk` and since then new solutions have been designed to overcome the limitations of partitions such as LVM and filesystem extents, which we will cover at the end of this chapter.

11.5 Logical Volume Manager

In order to enhance processing you may in your partitioning decisions want to place certain portions of the file-system on different disks. For instance you may want to place the `/var` directory on a different disk so that system log writing doesn't slow down data stored in the users home directories. You may be installing a MySQL database and want to move the default storage to a second disk you just mounted to reduce write ware on your hard disks. These are good strategies to employ, but what happens as the hard disks in those examples begin to fill up? How do you migrate or add larger disks?

The answer is that under standard partitioning and partitions you don't. You simply backup and reinstall the Operating System on a bigger drive. This is very time consuming and a risky operation that is not taken lightly. What to do? A solution to this problem and the limitations of traditional disk partitions is called LVM, [Logical Volume Management](#), created in 1998. LVM version 2 is the current full featured version baked in to the [Linux kernel since version 2.6](#).

LVM is a different way to look at partitions and file-systems. Instead of the standard way of partitioning up disks, instead we are dealing with multiple large disks. As technology progressed, we took our single large disk that we had split into partitions with `fdisk` and now we supplemented it with multiple disks in place of those partitions. The Linux kernel needed a new way to manage those multiple disks, especially in regards to a single file system. *“Logical volume management provides a higher-level view of the disk storage on a computer system than the traditional view of disks and partitions. This gives the system administrator much more flexibility in allocating storage to applications and users⁶.”* In order to install LVM2 on Fedora/CentOS and Ubuntu you can type:

- `sudo apt-get install lvm2`
- `sudo dnf install lvm2`
- `sudo yum install lvm2`

⁶<http://tldp.org/HOWTO/LVM-HOWTO/whatisvolman.html>

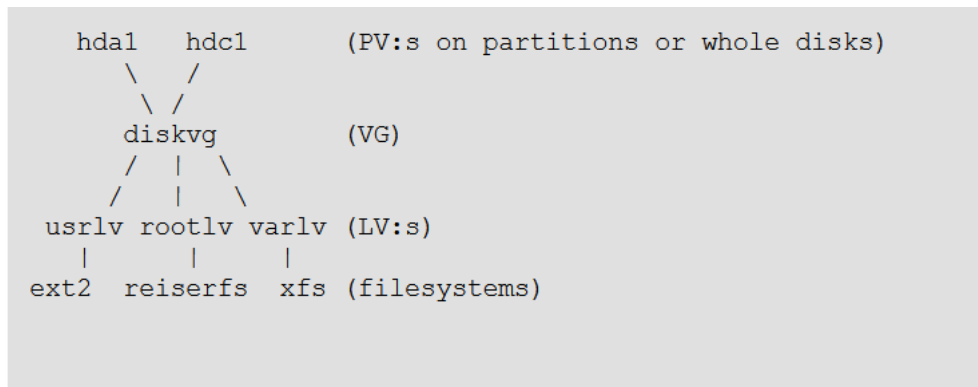


Figure 11.23: *LVM diagram*

This diagram creates three concepts to know when dealing with LVM:

- Volume Group (VG) - The Volume Group is the highest level abstraction used within the LVM. It gathers together a collection of Logical Volumes and Physical Volumes into one administrative unit ⁷.
- Physical Volume (PV) - A physical volume is typically a hard disk, though it may well just be a device that ‘looks’ like a hard disk (eg. a software raid device) ⁸.
- Logical Volume (LV) - The equivalent of a disk partition in a non-LVM system. The LV is visible as a standard block device; as such the LV can contain a file system (eg. /home) ⁹.
 - Physical Extent (PE) - This is the unit of storage (blocks) that a PV is split into
 - Logical Extent (LE) - This matches the PE and is used when multiple PVs are added to an LG, to make the *logical disk*. The LVM counts how many extents are possible and makes this its *disk* so to speak.

11.5.1 Physical Volumes

The first thing to do in creating an LVM partition is to figure out what kind of disks you have and what kind of partition scheme you want to use. Note that you can choose to use the entire disk `/dev/sdb` for instance or you can create a partition on the disk for use with LVM; if you do make sure to create the partition of type **0x8E** LVM and not a standard Linux partition. In order to use entire disks you need to use the `pvcreate` command to *create* physical volumes, same case with the partition.

11.5.2 Volume Groups

Once you have added the disks/partitions to the PV, now you need to create a Volume Group (VG) to add those PVs to. The command to add PVs to an LG is: `vgcreate VOLUME-GROUP-NAME /dev/sdx /dev/sdy`. You can extend this volume group by simply adding another `pvcreate /dev/sdx1` command for example and then using the `vgextend VOLUME-GROUP-NAME /dev/sdz`. There is also a `vgreduce` command that will remove a PV from a Volume Group. The volume group allows for a single logical management unit for multiple disk/partitions. This is useful as well for adding additional storage and removing storage devices that may have failed or are not performing at required parameters.

11.5.3 Logical Volumes

From within our Volume Group (VG) we can now carve out smaller LV (Logical Volumes). The nice part here is that the Logical Volumes don’t have to match any partition or disk size—since they are logically based on the combined size of the Volume Group which has extents mapped across those disks. Use the command `lvcreate -n LOGICAL-VOLUME-NAME --size 250G VOLUME-GROUP-TO-ATTACH-TO`. The `vgdisplay` command will show what had been created and what is attached to where. There are options to make the LV striped extents as opposed to linear, but that is an application based decision. Since LVs are logical they can also be extended and reduced on the fly—that alone is a better replacement

⁷<http://tldp.org/HOWTO/LVM-HOWTO/vg.html>

⁸<http://tldp.org/HOWTO/LVM-HOWTO/pv.html>

⁹<http://tldp.org/HOWTO/LVM-HOWTO/lv.html>

for standard partitioning. The command `lvextend -L 50G /dev/VOLUME-GROUP-NAME/LOGICAL-VOLUME-NAME` will extend the LV to become 50 GB in size. Using `-L+50G` will add 50 additional gigabytes to an existing LV's size.

Once you have successfully created an LV, now it needs a filesystem installed. Here you can add XFS, Ext4, Ext2, or any other file-system. You would use the `mkfs` proper tool for your filesystem. Once you have the filesystem created then you need a mount point. Each filesystem type has tools that allow you to extend the file-system automatically without the need to reformat the entire system, if the underlying LV or traditional partition is modified. Not all file-systems have the built in ability to shrink an existing partition.

11.5.4 LVM Snapshots

One definite feature not included in traditional partitioning is the concept of **snapshots**. **Snapshots** exist at the filesystem level in Btrfs and ZFS, but not XFS or ext4 as they are too old. The command `sudo lvcreate -s -n NAME-OF-SNAPSHOT -L 5g VOLUME-GROUP-NAME` creates a LV volume that is a snapshot or CoW, Copy-on-Write partition.

```
sudo lvcreate -L1000M -s -n lv-group-snapshot /dev/vg-group/lv-group
```

When dealing with LVM there is an ability to provide a snapshot, that is a point in time exact copy of a logical volume¹⁰. The snapshot can be a smaller disk size because this new LV is only going to copy the changes, or deltas, from the original LV, not duplicating data but sharing it between the two LVs. This delta can be merged back in, returning you to a point in time state, via the command:

```
sudo lvconvert --merge /dev/vg-group/lv-group-snapshot
```

Assuming you have your physical volumes, your volume groups, and logical volumes created, lets now create a snapshot of a logical volume (assume that we formatted the logical volume with ext4 and mounted it to `/mnt/disk1`).

```
# This works with ext4, but XFS requires two additional commands
# https://learn.redhat.com/t5/Platform-Linux/How-to-Mounting-an-XFS-LVM-snapshot/td-p/287
# xfs_repair and xfs_admin
cd /mnt/disk1
# command to create a random file of 5MB
head -c 5MB /dev/urandom > datafile.txt
ls -lh
# Picking up from the tutorial at http://tldp.org/HOWTO/LVM-HOWTO/snapshots_backup.html
sudo lvcreate -L1000M -s -n disk1snapshot /dev/volgroupname/logicalvolname
# Type the random command to change the data since the original snapshot
head -c 25MB /dev/urandom >> datafile.txt
# You will now have a new device called /dev/volgroupname/disk-backup
# Mount this to /mnt/disk2
sudo mkdir -p /mnt/disk2
sudo mount -t ext4 /dev/volgroupname/disk1snapshot /mnt/disk2
cd /mnt/disk2
ls -l
ls -l /mnt/disk1
# What do you see? Why?
```

You can remove the snapshots by unmounting the partition `umount` and then using the `lvremove` command. How would you do this same process using ext4 without LVM?

11.6 Filesystems

To extend our analogy of a disk drive being like land, and a partition being like different lots of land sold off to different people, then a filesystem would be the actual building that is built on the property to make use of the land, be it farm land, nature preserve, solar plant, or factory. A **filesystem** is the way that an operating system addresses, stores, and retrieves data stored on a disk. It is an in-between layer so the operating system can have an addressing scheme for data, without having to know the exact mapping of the particular disk drive in question.

¹⁰<http://tldp.org/HOWTO/LVM-HOWTO/snapshotintro.html>

If you have used Windows before you are familiar with FAT32 and NTFS filesystems. Since Windows is created and curated by Microsoft, there has only been two different filesystems in the history of Windows. Linux on the other-hand supports multiple different filesystems that serve many different purposes.

11.6.1 ext/ext2

The MINIX filesystem was the first Linux based filesystem released in 1991. It was borrowed conceptually from the Minix operating system that Andrew Tanenbaum had created. It had severe limitations since MINIX filesystem was engineered to be *ultra* backwards compatible—hence had 16 bit offsets and had a maximum partition size of 64 megabytes. By 1992 and Linux 0.96c a new filesystem replacement called **ext** was created and brought into Linux as the native filesystem. By January of 1993, **ext2** had been created and additional features added, including future proofing the system by adding unused options that could later on be tested and added as need arose. Like most operating systems, data is broken up into **blocks**, which is the smallest sized piece of data that can be read or written by the operating system¹¹.

Table 11.3: Limits of ext2

Block size:	1 KiB	2 KiB	4 KiB	8 KiB
max. file size:	16 GiB	256 GiB	2 TiB	2 TiB
max. filesystem size:	4 TiB	8 TiB	16 TiB	32 TiB

Traditionally your `/boot` partition is formatted as **ext2** because it is only used for a short time to load your *initrd* and *kernel image* into memory, so the overhead of **ext4** is not needed. You can use the built in `sudo mkfs` command to format a partition with **ext2**.

11.6.2 ext3/ext4

By 2001, as filesystems became larger, the amount of data being written increased and the chances for data corruption or disk writes to fail became more evident and critical. The CPU could now handle to overhead of managing data writes to disk to ensure that those operations actually happened.

“A journaling file system is a file system that keeps track of changes not yet committed to the file system’s main part by recording the intentions of such changes in a data structure known as a “journal”, which is usually a circular log. In the event of a system crash or power failure, such file systems can be brought back online quicker with lower likelihood of becoming corrupted¹²¹³.”

Not to be confused with `journald` from `systemd`, the **ext3** filesystem, introduced to the Linux kernel the journaling feature in 2001. Being an extension basically of **ext2**, **ext3** began to inherit legacy problems of **ext** and **ext2** as they were now over a decade old.

Table 11.4: Limits of ext3

Block size	Max file size	Max file system size
1 KiB	16 GiB	4 TiB
2 KiB	256 GiB	8 TiB
4 KiB	2 TiB	16 TiB

By 2008 it became apparent that **ext3** has reached the end of its development, and [Theodore Ts'o](#) announced that **ext4** would extend the **ext** filesystem a bit longer, but the growth of **ext** had hit the end, and a newer filesystem needed to be developed to handle the larger sets of data and the massively improved hardware that existed since 1992, when **ext** was developed. A major drawback of **ext4** is the lack of snapshot support (like LVM) and the need to be backward compatible with **ext2**.

¹¹<https://en.wikipedia.org/wiki/Ext2>

¹²https://en.wikipedia.org/wiki/Journaling_file_system

¹³<https://en.wikipedia.org/wiki/Ext3>

Ext4 saw the capacity extension of **ext3** and introduction to **extents**. The **ext4** filesystem can support volumes with sizes up to 1 exibyte (EiB) and files with sizes up to 16 tebibytes (TiB).

In ext4, **extents** replaced the traditional block mapping scheme used by ext2 and ext3. An extent is a range of contiguous physical blocks, improving large file performance and reducing fragmentation. A single extent in **ext4** can map up to 128 MiB of contiguous space with a 4 KiB block size ¹⁴. The advantage is that the smaller 4KB blocks are bound contiguously on disk together.

Theodore Ts'o is a respected developer in the open source community, who currently is the maintainer of **ext4** and is employed by Google to develop filesystems. **Ext4** is the current default file system for most Linux distros, though that is changing as OpenSuse and Fedora 33 have adopted using Btrfs and Ubuntu is working on adopting OpenZFS as their default filesystem. The advantage of ext4 is it is well tested and a well known quantity and is currently used by Google in Android devices as well. To format a partition using the **ext4** filesystem you would simply type `mkfs.ext4` and the partition will be formatted. You normally don't format entire devices, just partitions, which can take up entire disks. There are three additional competing filesystems since 2008 that fill the void left by **ext4**.

11.6.3 Giga vs Gibi

These matter because humans and computers (and marketing) use different numbering systems.

Table 11.5: Multiples of bytes

Metric	IEC	JEDEC
1000 kB kilobyte	1024 KiB kibibyte	KB kilobyte
1000 MB megabyte	1024 MiB mebibyte	MB megabyte
1000 GB gigabyte	1024 GiB gibibyte	GB gigabyte
1000 TB terabyte	1024 TiB tebibyte	
1000 PB petabyte	1024 PiB pebibyte	
1000 EB exabyte	1024 EiB exbibyte	
1000 ZB zettabyte	1024 ZiB zebibyte	
1000 YB yottabyte	1024 YiB yobibyte	

11.6.4 XFS

XFS is a robust and highly-scalable single host 64-bit journaling file system. It is entirely extent-based, so it supports very large files and file system sizes. The maximum supported file system size is 100 TB. The number of files an XFS system can hold is limited only by the space available in the file system ¹⁵.

XFS was originally created by SGI (Silicon Graphics Inc) back in 1993 to be a high-end Unix work station filesystem. SGI was the company that made computers in the 1990's for high end movie special effects and graphical simulation. They had their own version of Unix called IRIX, and needed a filesystem capable of handling large files at that time, and places like NASA which had large amounts of data to store and access. SGI created XFS to suit that need. XFS excels in the execution of parallel input/output (I/O) operations due to its design, which is based on allocation groups (a type of subdivision of the physical volumes in which XFS is used-also shortened to AGs). Because of this, XFS enables extreme scalability of I/O threads, file system bandwidth, and size of files and of the file system itself when spanning multiple physical storage devices ¹⁶.

XFS was ported to Linux in 2001 as SGI and IRIX went out of business and the filesystem languished. It was opensourced and GPL'd in 2002. Red Hat began to see this filesystem as an alternative to ext4 and more mature than other replacements since it had over 10 years of development from the start to handle large sized files. Red Hat also hired many of the SGI engineers and developers who created this filesystem and brought it back into production quality. Red Hat began with RHEL 7 to deprecate ext4 as the default filesystem and implement XFS as their standard filesystem on the RHEL product.

¹⁴<https://en.wikipedia.org/wiki/Ext4>

¹⁵https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Performance_Tuning_Guide/s-storage-xfs.html

¹⁶https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Performance_Tuning_Guide/s-storage-xfs.html

XFS is notoriously bad at being used by an everyday computer because its strength is build on using a system storing large database files or archiving large files. You can install the tools needed to make a partition of the XFS format by typing `sudo apt-get install xfsprogs`; the XFS tools are already installed on Fedora and CentOS by default. You can create an XFS filesystem using the `sudo mkfs.xfs` command. We can grow an XFS filesystem with the command `xfs_growfs /mount/point -D size`.

11.6.5 Next Generation Linux Filesystems

The ext4 filesystem served its purpose well but by 2008 became apparent that ext4 was not the right filesystem design for taking full advantage of memory, disk, and processor improvements—as well as the changing use case of computing focusing on large dynamic clusters such as service companies like Google, Facebook, Twitter, and other social media companies. These new generation of filesystems combine the filesystem, volume management, data compression, volume snapshots, and datafile integrity. The two main candidates that are opensource and designed to take advantage of this current technological environment are [Btrfs](#) and [OpenZFS](#).

11.6.6 Btrfs

This project was initially created by Chris Mason at Oracle in 2007, for use on their own storage products to compete against SUN Microsystems ZFS filesystem. By 2013 it was considered stable and included in the Linux Kernel. Facebook is currently where Chris Mason is employed and they are championing the use of this operating system on [their infrastructure](#).

Btrfs is a modern copy-on-write (CoW) filesystem for Linux. Copy-on-write is at its core and optimization pattern that uses pointers instead of making multiple copies of data on a disk, therefore reducing write operations. When the original file is modified then a true on disk copy of the file is made¹⁷. Chris Mason said the goal of Btrfs is, *“to let Linux scale for the storage that will be available. Scaling is not just about addressing the storage but also means being able to administer and to manage it with a clean interface that lets people see what’s being used and makes it more reliable”*¹⁸.

Btrfs adds support for resource pooling and using extents to make logical drives across physical devices removing the need for the use of LVM, volume management is now built in. Recently openSUSE and Fedora have adopted Btrfs as a filesystem, but support for Btrfs was remove in RHEL 8 (in favor of XFS and LVM).

In order to format a system using Btrfs you need to install `btrfs-progs` on Fedora 32+ and Ubuntu 20.04.

11.7 Install Btrfs tools

```
sudo dnf install btrfs-progs    sudo yum install btrfs-progs    sudo apt-get install btrfs-tools -
Ubuntu 18.04 sudo apt-get install btrfs-progs - Ubuntu 20.04
```

Table: Demonstration of Btrfs syntax

11.7.1 Btrfs Creation Commands

- Create a btrfs file system on a single device. For example:
 - `mkfs.btrfs /dev/sdb1`
- Create a btrfs file system with a label that you can use when mounting the file system. For example:
 - `mkfs.btrfs -L myvolume /dev/sdb2`
 - Note: The device must correspond to a partition if you intend to mount it by specifying the name of its label.
- Create a btrfs file system on a single device, but do not duplicate the metadata on that device. For example:
 - `mkfs.btrfs -m single /dev/sdc`
- Stripe the file system data and mirror the file system metadata across several devices. For example:
 - `mkfs.btrfs /dev/sdd /dev/sde`
- Stripe both the file system data and metadata across several devices. For example:
 - `mkfs.btrfs -m raid0 /dev/sdd /dev/sde`

¹⁷https://btrfs.wiki.kernel.org/index.php/Main_Page

¹⁸<https://en.wikipedia.org/wiki/Btrfs>

- Mirror both the file system data and metadata across several devices. For example:
 - `mkfs.btrfs -d raid1 /dev/sdd /dev/sde`
- Stripe the file system data and metadata across several mirrored devices. You must specify an even number of devices, of which there must be at least four. For example:
 - `mkfs.btrfs -d raid10 -m raid10 /dev/sdf /dev/sdg /dev/sdh /dev/sdi /dev/sdj /dev/sdk`

Btrfs extensive documentation can be found at Oracle’s website: https://docs.oracle.com/cd/E37670_01/E37355/html/ol_create_btrfs.html

11.7.2 ZFS

ZFS is a filesystem originally developed by Sun for their Solaris Unix operating system and in use since 2001. ZFS was opensourced by SUN in 2005 under the CDDL license (similar to the Mozilla Public License) but incompatible with the GPL. Oracle inherited ZFS when it invaded SUN in 2010. It is not licensed under the GPL but under a Sun/Oracle license called **CDDL**, which is similar to the GPL, but allowed Sun and Oracle to include proprietary parts of the operating system with opensource code. The CDDL is an opensource license, but not a copy-left license and thus GPL incompatible. Recently Canonical, Ubuntu’s parent company, disagreed with this, and began to offer OpenZFS natively as part of their operating system. The argument of integrating CDDL based ZFS code into GPLv2 Linux Kernel was extensive with the FSF coming down in opposition of Ubuntu’s interpretation of the GPL.

- [GPL Violations Related to Combining ZFS and Linux](#)
- [Interpreting, enforcing and changing the GNU GPL, as applied to combining Linux and ZFS](#)
- [Ubuntu ZFS announcement](#)

FreeBSD didn’t have this restriction under the BSD license and they have had native kernel based support for ZFS since version 9 of FreeBSD and ZFS is a supported filesystem type on MacOS. As of Ubuntu 16.04, you can install ZFS via apt-get and include the CDDL licensed ZFS code on Linux as a loadable kernel module. Ubuntu now supports the root partition being ZFS as well.

Development of all ZFS code now lives in the upstream [OpenZFS project](#). Since the ZFS code was opensourced, when Oracle tried to “closesource” the code base in 2010, essentially what Oracle did was make a fork of the project and keep their changes proprietary. The rest of the community took ZFS and made the OpenZFS community, which now consolidates various ZFS code bases into a single repo for MacOS, FreeBSD, and Linux as a loadable kernel module. RedHat has not participated in the Linux development of OpenZFS as most companies are afraid of potential litigation and lawsuits from Oracle (real or perceived). There is even a ZFS developer port who brought [ZFS to Windows](#), using the latest Windows 10 OS.

ZFS is an elegantly designed filesystem: *“ZFS is a combined file system and logical volume manager designed by Sun Microsystems. The features of ZFS include protection against data corruption, support for high storage capacities, efficient data compression, integration of the concepts of filesystem and volume management, snapshots and copy-on-write clones, continuous integrity checking and automatic repair, Software based RAID, (RAID-Z)¹⁹.”*

11.7.2.1 ZFS Installation

Here is an example to install the ZFS module, load the module, and then format and create a zpool logical mirror (RAID1) in a few steps, tutorial comes from here: <https://wiki.ubuntu.com/Kernel/Reference/ZFS>.

```
sudo apt install zfsutils-linux

# Now check to see if the zfs module is loaded
modprobe zfs
lsmod | grep zfs
```

11.7.3 ZFS RAID-Z

ZFS supports creating multiple types of redundant disks that increase speed or increase redundancy, such as:

- ZFS stripe
 - Two or more disks that are collected into a single logical disks in order to speed up writes at the expense of redundancy

¹⁹<https://en.wikipedia.org/wiki/ZFS>

- Also known as a [RAID 0](#)
- ZFS mirror
 - Disks in pairs of 2 that mirror each other's writes for fail over protection in case of disk failure
 - Also known as a [RAID 1](#)
- ZFS striped mirror
 - Two or more disks that are striped and then mirrored for redundancy but also gains the advantage of the stripes speed
 - Also known as a RAID 10
- ZFS RAID-Z
 - Using multiple disks and redundancy to support multiple disk failure and recovery
 - Also known as a [RAID 5](#)

```
# Assume each disk is 2 GB in size
# Create a Stripe of 3 disks for a single 6 GB logical volume
sudo zpool create datastripe /dev/sdb /dev/sdc /dev/sdc
sudo zpool status
```

```
# Create a Mirror of 2 disks for a single 2 GB logical volume
# with failover protection
sudo zpool create datamirror mirror /dev/sde /dev/sdf
sudo zpool status
```

```
# Create a Striped Mirror of 4 disks for a logical volume of 4 GB
# with failover protection
sudo zpool create sm mirror /dev/sdg /dev/sdh /dev/sdi /dev/sdj
sudo zpool status
```

11.7.4 ZFS Snapshots

ZFS has support for snapshots similar to LVM but built into the filesystem layer. Let's demonstrate this by creating a file named **accounts.txt** on /datamirror.

```
# Create the mirror -- change the devices names as needed
sudo zpool create datamirror mirror /dev/sde /dev/sdf

# Change the ownership to of the datamirror pool
# Assume your user name is controller
sudo chown -R controller:controller /datamirror

# Create a file of 25 mb size
cd /datamirror
truncate -s 25m accounts.txt
sudo zfs snapshot datamirror@snap-datamirror1
sudo zfs list -t datamirror@snap-datamirror1

# Lets modify the contents of datamirror and then rollback the snapshot
truncate -s 50m new-accounts.txt
touch newer-accounts.txt
ls
# Lets rollback the changes to pre-snapshot
sudo zfs rollback datamirror@snap-datamirror1
# Issue the ls command and you will see all the additional files gone
ls
```

11.7.5 ZFS Send and Receive

ZFS also has a mechanism to send and receive snapshots, which done in a small enough increments which effectively creates a serialized synchronization feature. This can be done on the same system as well as over a network connection to a remote computer. To synchronize a ZFS filesystem:

- First create a snapshot of a zpool
- Using the `zfs send` and `zfs receive` commands via a pipe you can send your snapshot to become another partition
 - `zfs send datapool@today | zfs recv backuppool/backup`
- You can pipe the command over `ssh` to restore to a remote system
 - `zfs send datapool@today | ssh user@hostname sudo zfs recv backuppool/backup`

11.7.5.1 Additional ZFS Features

In addition there is an L2ARC cache for caching most recent and most frequently used data blocks. This is a separate SSD based disk and can speed up data access^{20 21}. The ZIL and the L2ARC if not defined on separate disks will take a small portion of the each zpool created. This is fine for low volume disk writes, but puts extra overhead on the system. If you have fast SSDs that are small in size, say 30 to 80 GB, you can stripe them and place the L2ARC cache and or ZIL on these disks. ZIL is a write only function and L2ARC cache becomes read predominant.

11.7.5.2 ZFS ZIL and SLOG

ZFS shines by creating additional read and write caches, called a ZIL and a SLOG. These caches enable the disks to reach and write in consistent batches or blocks of data instead of small random amounts of data. This way the disk can “report” a successful write, but the data is actually momentarily cached – then flushed to the disk along with block of writes.

Using an existing zpool, called **datapool**, we can attach two additional disk `/dev/sde` and `/dev/sdf`. You can add the directives for the log and cache after the `zpool create` command: `zpool add datapool cache /dev/sde2 /dev/sdf2 log mirror /dev/sde1 /dev/sdf1`. You can use the `/dev/` locations of disks or you can see your UUIDs with the `blkid` or `lblkid --fs` command²².

“ZFS Intent Log, or ZIL- A logging mechanism where all of the data to be the written is stored, then later flushed as a transactional write. Similar in function to a journal for journaled filesystems, like ext3 or ext4. Typically stored on platter disk. Consists of a ZIL header, which points to a list of records, ZIL blocks and a ZIL trailer. The ZIL behaves differently for different writes. For writes smaller than 64KB (by default), the ZIL stores the write data. For writes larger, the write is not stored in the ZIL, and the ZIL maintains pointers to the synched data that is stored in the log record²³”.

“Separate Intent Log, or SLOG- A separate logging device that caches the synchronous parts of the ZIL before flushing them to slower disk. This would either be a battery-backed DRAM drive or a fast SSD. The SLOG only caches synchronous data, and does not cache asynchronous data. Asynchronous data will flush directly to spinning disk. Further, blocks are written a block-at-a-time, rather than as simultaneous transactions to the SLOG. If the SLOG exists, the ZIL will be moved to it rather than residing on platter disk. Everything in the SLOG will always be in system memory²⁴”.

11.7.5.3 ZFS Disk Scrubbing

ZFS supports disk scrubbing. Which will check every block of data against its own checksum meta-data and clean up any silent corruption. ZFS has a known good list of checksums of all blocks of data, and is constantly watching for corruption of data. Scrubs do not happen automatically but can be scheduled to run periodically. You can check the status of a disk with the command `zpool status datapool` and execute a scrub command `zpool scrub datapool`.

²⁰<https://pthree.org/2012/12/07/zfs-administration-part-iv-the-adjustable-replacement-cache/>

²¹<http://www.c0t0d0s0.org/archives/5329-Some-insight-into-the-read-cache-of-ZFS-or-The-ARC.html>

²²<https://pthree.org/2012/12/07/zfs-administration-part-iv-the-adjustable-replacement-cache/>

²³<https://pthree.org/2012/12/06/zfs-administration-part-iii-the-zfs-intent-log>

²⁴<https://pthree.org/2012/12/06/zfs-administration-part-iii-the-zfs-intent-log>

11.7.5.4 ZFS Transparent Disk Compression

ZFS can enable transparent compression using GZIP or LZ4 with a simple set command: `zfs set compression=lz4 datapool`. This can help and there is little overhead. Finally ZFS supports data-deduplication on a file basis. If enabled each file is hashed with sha-256 and any files that match, only 1 of the files is kept, the others have markers pointing back to this original file. This saves the overall amount of data you are storing and can reduce costs but the cost is high in amount of ram needed to store the de-dupe tables. You can display the current compression level with the command: `sudo zfs get compression mydatapool`.

11.7.5.5 Finding a physical disk



ZFS, Btrfs, and LVM have the ability to remove disks from pools and volumes. The trouble is you can remove the disk logically—but how do you identify which physical disk it is? Luckily each disk has a serial number printed on the top of it. When working in these scenarios you should have all of these serial numbers written down as well as the location of where that disk is. You can find the serial number of the disk via the `hdparm` tool. This script would enumerate through all of the disks you have on a system and print the values out. Note the a, b, c, are a list of the device names. In this case there is hard drive `/dev/sda` through `/dev/sdg`²⁵ run the command on your system and see what comes out.

```
for i in a b c d e f g;
do
    echo -n "/dev/sd$i: "
    hdparm -I /dev/sd$i | awk '/Serial Number/ {print $3}'
done
```

#If you are missing those tools, just install following packages

```
sudo apt-get install hdparm
sudo apt-get install smartmontools
sudo apt-get install lshw
```

```
# These commands all will show you information relating to the disk serial number
# https://unix.stackexchange.com/questions/121757/harddisk-serial-number-from-terminal
sudo lshw -class disk
sudo smartctl -i /dev/sda
lsblk --nodeps -o name,serial
```

11.7.6 HFS+, UFS, and APFS

The BSD systems has its own filesystem, UFS: [the Unix File System](#). This filesystem was native to Unix going back to the System 7 Unix release. Though UFS has been updated since and is officially UFS2, which was released around 1994 when BSD split from Unix due to the AT&T lawsuit. UFS2 is considered similar to ext4 on Linux in capabilities at the current time. FreeBSD and then other BSDs adopted ZFS to be able to extend the filesystem, capability of UFS. All Illumos, or OpenSolaris based distros use ZFS natively as well, but [BSD based systems have switched](#) their ZFS to be based on the [ZFS on Linux](#) code base due to a larger developer and feature base.

Apple had been using their own filesystem called HFS+ which was introduced in 1998 in MacOS 8.1 in 1998. Features were added over time and in each release to keep the filesystem with feature parity for ext4. This was used as the standard file system on all Mac and iOS devices until 2016/2017 when a new Apple designed filesystem was released across devices. Why a new filesystem? Think about it, by 2016 what was the primary device that Apple was selling

²⁵CC BY-SA 3.0 By Redline from Wikimedia Commons

²⁶<https://pthree.org/2012/12/11/zfs-administration-part-vi-scrub-and-resilver/>

compared to 1998? What type of storage media had HFS+ been designed for? What type of storage media was running on the new Apple systems, in laptops and mobile?

Though Apple Was one of the first companies to [port ZFS to MacOS](#), which is BSD based, eventually they decided to deploy a new filesystem called APFS (apple filesystem) which has a mix of ZFS like features that were home grown by Apple and more importantly not under the control of another company or any particular free or opensource licenses. The goal for Apple was to use this filesystem across their platform, from MacOS to iOS devices with a focus on Flash bashed (SSD and NVMe) devices that HFS+ wasn't designed for. The system has a similar feature set to ZFs and Btrfs such as encryption and snapshots, but [is missing data integrity](#).

11.7.7 F2FS and EROFS

Due to the increased use of NAND or Flash based memory storage such as SSD, eMMC, and SD cards, a consortium of companies dealing with mobile devices came together in 2012 to create a filesystem which takes the nature of Flash into direct account in order to overcome some of the weaknesses of Flash based storage. F2FS stands for Flash-Friendly File System, and was developed by Samsung Electronics, Motorola Mobility, Huawei and Google. The F2FS systems was adopted for some devices in 2012-2016 and saw a resurgence in use by 2018 led by the Pixel 3 and Motorola devices.

Hauwei replaced F2FS on their Honor devices with the [Enhanced Read-Only File System](#), or EROFS. It is designed and maintained by Huawei and is used on their own Android based devices introduced in 2018 and is part of the Linux Kernel.

11.7.8 DragonFly BSD and Hammer FS

DragonFly BSD developer Matthew Dillion has been spearheading the development of his own distributed cluster based filesystem called [Hammer](#). His goal is to have finer grained snapshotting—even per file on a constant basis and make snapshots almost a constant occurrence across the filesystem for easy migration of a filesystem to a different Hammer Cluster (migration over a network). Work has recently finished on the Hammer 2 file system which is now an option for installation on DragonFly BSD 5.2+. The Clustering feature is still a work in progress though, but Hammer has similar ZFS and Btrfs style features of file system and volume-management. HammerFS is DragonFly BSD code, though it could be ported to other BSDs, the DragonFly internals have diverged so far that this becomes essentially impossible.

11.8 Mounting and Unmounting of disks

Once a disk is partitioned, and formatted with a filesystem, it now needs to be mounted. The concept of mounting came from the UNIX days of carrying a large reel of magnetic tape, and physically mounting it on a tape reader. You can see all the mount points currently attached to your system by typing `/etc/mtab`. A filesystem needs to be mounted to a directory location. Technically your root filesystem is mounted to the `/` partition.

In the previous examples we we have created partitions and filesystem, now let us mount them. The first step we need to do is provide a mount point. Traditionally that is done in the `/mnt` directory. You should create your **mountpoints** here. Let's type `sudo mkdir -p /mnt/data-drive`. The name *data-drive* is an arbitrary name I have given my newly created **mountpoint**. The `-p` flag will auto-create any subdirectory under `/mnt` that doesn't already exist. Why did I type `sudo`? Who owns the `/mnt` directory?

Once this directory is created, you can use the `mount` command like this: `sudo mount -t ext4 /dev/sdb /mnt/data-drive`. The `-t` flag tells this mount that the filesystem is of type `ext4` and the operating system needs to know so that it can interface correctly with the filesystem. Once this is done, the directory will still be owned by root, you probably need to change the ownership of the directory so that you own and can write to it. How would you do that based on last chapter? You could type `sudo chown controller:controller /mnt/data-drive`, assuming your username is *controller*.

The partition can be unmounted by typing the `umount` command—yes it is missing the `n`. Be careful you don't try to unmount the device while your `pwd` is in a directory on that mount—otherwise you will get a *device is busy error*.

11.8.1 /etc/fstab

The `/etc/fstab` file controls the automatic mounting of your filesystems at boot. Every time your system boots, technically each partition is remounted every time too. If you create your own filesystem and want it mounted automatically on boot, then you would need to add an entry here. The `/etc/fstab` file has 6 columns containing values listed here: `<device> <mount point> <fs type> <options> <dump> <pass>`.

An example entry could contain these values: `/dev/sdb1 /mnt/data-drive ext4 defaults 0 0`. Devices now are typically listed by their UUID, which can be found by typing `ls -l /dev/disk/by-uuid` or `lsblk --fs /dev/sda`. That is the actual command not a place holder. This is where the long strings you see in the `/etc/fstab` file in place of the device name.

The use of UUIDs is a better idea as once a UUID is given, even if the device name changes the UUID will not. Here is a sample `/etc/fstab` file showing UUID based fstab with an additional disk being mounted.

```
# <file system> <mount point> <type> <options> <dump> <pass>
# / was on /dev/sda1 during installation
UUID=e720a798-b02c-4e3f-8132-8f67f2be0c2c /          ext4      errors=remount-ro 0 1
/swapfile                               none      swap      sw              0 0
UUID=003e6f67-9d31-4198-b3dd-4447f2337445 /mnt/disk2 btrfs     defaults   0 0
```

There are many options that can be set in the place of `defaults` or appended via a “,” such as:

1. `sync/async` - All I/O to the file system should be done (a)synchronously.
2. `auto` - The filesystem can be mounted automatically (at bootup, or when mount is passed the `-a` option). This is really unnecessary as this is the default action of `mount -a` anyway.
3. `noauto` - The filesystem will NOT be automatically mounted at startup, or when mount passed `-a`. You must explicitly mount the filesystem.
4. `dev/nodev` - Interpret/Do not interpret character or block special devices on the file system.
5. `exec / noexec` - Permit/Prevent the execution of binaries from the filesystem.
6. `suid/nosuid` - Permit/Block the operation of `suid`, and `sgid` bits.
7. `ro` - Mount read-only.
8. `rw` - Mount read-write.
9. `user` - Permit any user to mount the filesystem. This automatically implies `noexec`, `nosuid`, `nodev` unless overridden.
10. `nouser` - Only permit root to mount the filesystem. This is also a default setting.
11. `defaults` - Use default settings. Equivalent to `rw`, `suid`, `dev`, `exec`, `auto`, `nouser`, `async`.
12. `netdev` - this is a network device, mount it after bringing up the network. Only valid with `fstype nfs`.

11.8.2 systemd Mounting Units

Usually `systemd` will strive to absorb functions, but according to the man page for `systemd.mount`, “Mounts listed in `/etc/fstab` will be converted into native units dynamically at boot and when the configuration of the system manager is reloaded. In general, configuring mount points through `/etc/fstab` is the preferred approach. See `systemd-fstab-generator(8)` for details about the conversion.” ZFS will take care of automounting its own partitions. Btrfs you will need to add an entry using the UUID instead of the `dev` name which you can find via the `blkid` command.

`Systemd` has absorbed the purpose of the `/etc/fstab` file by creating `.mount` files. `Systemd` processes `.mount` files first then processes the `/etc/fstab` file. Let’s look at an example where we have already created a Btrfs filesystem on a new virtual disk. By using the `lsblk --fs` command we can retrieve the UUID of the disk. The reason we would want look at using the UUID instead of the `/dev/sdx` is that once assigned the UUID won’t change. The device designation can change. Say for instance that a hard drive fails, on reboot your system will re-enumerate the devices and now hard drive names will be changed.

In this example the output of the command, `lsblk --fs` will look like this (assuming you have attached the virtual disk and formatted it as Btrfs):

```
sda
sda1  ext4      e720a798-b02c-4e3f-8132-8f67f2be0c2c /
sdb   btrfs     003e6f67-9d31-4198-b3dd-4447f2337445
sdc   btrfs     62bab009-e64e-445c-bf33-5f2143569a83
```



```
# Create this file: /etc/systemd/system/mnt-databasedisk.mount
# Note that the .mount file name needs to be the same as the "Where" location
# Replace the / with -
# Make sure to enable the .mount at boot
# sudo systemctl enable mnt-databasedisk.mount
```

```
[Unit]
Description=Disk that was added to host the storage of our database
```

```
[Mount]
What=/dev/disk/by-uuid/003e6f67-9d31-4198-b3dd-4447f2337445
Where=/mnt/databasedisk
Type=btrfs
Options=defaults
```

```
[Install]
WantedBy=multi-user.target
```

Remember! ZFS handles explicit mounting of volumes/zpools as part of the act of creating zpools and do not need to have `.mount` files created.

11.8.3 Disk related tools

There are two useful commands to use in regards to understanding the disk resource use in regards to the filesystem. The `df` command will list the disk usage. There is an optional `-H` and `-h` which presents the file-system usage in Gigabytes (`-H` is metric: giga, `-h` is binary, gibi). When you use `df` without any directories, it will list all file-systems. The command below lists the file-system that contains the user's home directory: `/home/controller` for example.

```
[controller@localhost ~]$ man df
[controller@localhost ~]$ df -H /home/controller
Filesystem      Size  Used Avail Use% Mounted on
/dev/mapper/fedora-root 17G  7.7G  8.0G  50% /
[controller@localhost ~]$
```

Figure 11.24: `df -H /home/controller`

The `du` command is disk usage. This is a helpful command to show the exact *byte-count* that each file is actually using. When using `ls -l` Linux reports only 4096 kb for a directories size, this does not actually reflect the size of the content inside the directory. The `du` command will do that for you.

These commands might not report completely accurate information when dealing with next generation filesystems like ZFS and Btrfs. use the commands: `sudo btrfs filesystem usage [mountpoint]` or `sudo zfs list [poolname]`

```
sudo btrfs filesystem usage /mnt/databasedisk
Overall:
  Device size:          3.00GiB
  Device allocated:    331.12MiB
  Device unallocated:  2.68GiB
  Device missing:      0.00B
  Used:                 320.00KiB
  Free (estimated):    2.68GiB (min: 1.35GiB)
  Data ratio:           1.00
  Metadata ratio:      2.00
  Global reserve:      3.25MiB (used: 0.00B)

Data,single: Size:8.00MiB, Used:64.00KiB
  /dev/sdc    8.00MiB

Metadata,DUP: Size:153.56MiB, Used:112.00KiB
  /dev/sdc   307.12MiB
```

```
System,DUP: Size:8.00MiB, Used:16.00KiB
/dev/sdc 16.00MiB
```

```
Unallocated:
/dev/sdc 2.68GiB
```

When you compare the space consumption that is reported by the `df` command with the `zfs list` command, consider that `df` is reporting the pool size and not just file system sizes. In addition, `df` doesn't understand descendent file systems or whether snapshots exist. If any ZFS properties, such as compression and quotas, are set on file systems, reconciling the space consumption that is reported by `df` might be difficult²⁷.

```
# df -H /data1
Filesystem      Size  Used Avail Use% Mounted on
data1           1.9G  132k  1.9G   1% /data1
```

```
# sudo zfs list data1
NAME      USED  AVAIL    REFER  MOUNTPOINT
data1    104K  1.75G      24K   /data1
```

11.9 Compression and Archiving tools

If you remember the history of Unix and history of technology you remember that hard drive space was at a premium for many many years. Also the concept of discrete hard drives did not come about until the early to mid 1980s. Things we take for granted now—such as zipping a series of folders and attaching them as a single file in email were unthinkable in 1979.

11.9.1 tar

By 1979 local storage had increased to the point where it was conceivable that a **tar** archive or tar file could be taken of a directory structure for backup purposes preserve the directory structure. The **tar** command was created and first included in Unix System 7 release. The added advantage was that the tar file could be transferred as a single file, thereby reducing network overhead and os seek-time but retain a hierarchy of directories. This method also became the preferred way to distribute code that was used to compile applications. One could just un-tar an archive and then compile the code knowing that the directory structure of the included files was correctly preserved.

The tar command only does archiving and does not do any compression—only preserving of file structure in the same way that an ISO file preserves structure. The tar archive by convention is assigned a file extension of **.tar** but this is not added automatically.

Example Usage: `tar -cvf code.tar ./code-directory` This command will create a **tar** archive of the directory called code-directory. `tar [options] [archive name and location] [what to archive]`

Example Usage: `tar -xvf code.tar` This command will unpack or extract a **tar** archive of the directory called code-directory and place it in the `pwd`.

11.9.2 compress

As file sizes grew the need to compress redundant data became apparent. In dealing with compression you have two sides and you have to choose one. Either the fast time to compress and larger file sizes, or slower time to compress and smaller file sizes. The initial compression algorithms went for the faster compression but larger file option. The first compression tool on Unix was called **compress**. But it was encumbered by a patent on the **LZW** compression algorithm, the same patent on that lead to the creation of the jpeg image standard to replace the encumbered GIF image format. Because of this compress was never *free* and outside of its invention and inclusion in commercial Unix in 1985, use could never catch on.

²⁷https://docs.oracle.com/cd/E23824_01/html/821-1448/gbchp.html

Compress/Uncompress is a Unix shell compression program based on the LZW compression algorithm. Compared to more modern compression utilities such as gzip and bzip2, compress performs faster and with less memory usage, at the cost of a significantly lower compression ratio ²⁸.

11.9.3 gzip

By 1991, Phil Katz had created an opensource implementation of LZW called [DEFLATE](#). This was the basis of the popular PKZip program and the origin of the .zip compression extension. The [GNU project](#) began its own GPL based implementation using the DEFLATE algorithm and completed it by October of 1992. It was named gzip ([GNU zip](#)).

gzip is based on the DEFLATE algorithm, which is a combination of LZ77 and Huffman coding. DEFLATE was intended as a replacement for LZW and other patent-encumbered data compression algorithms which, at the time, limited the usability of compress and other popular archivers ²⁹.

11.9.4 bzip2

This is a similar algorithm to gzip in that they do compression only. It differs from gzip in that it uses the [LZMA](#) compression algorithm which produces smaller sized files but take more CPU time and memory to compress. This is deemed an acceptable trade off as computers are only getting faster. Decompression is very fast compared to compression. It was released in 2001³⁰ and is available under a BSD-like license.

11.9.5 xz

The [xz](#) compression tool is using the [LZMA2](#) compression algorithm which is superior in decreasing size at the expense of compute time. The xz algorithm uses LZMA2 which has support for multithreading in compressing and decompressing in parallel. Back in 1985 when the first Compress program was created processing power was slow compared to the speed of the networks. Now the speed of the processor is much faster relative to the speed of our networks. On today's networks we are moving multiple gigabytes around at a time. The xz compression tool is the tool for the future. In December 2013, [kernel.org](#) announced the addition of xz compressed files and ending bzip2 compressed files for distributing the Linux kernel archive files. The xz tool works only on single files and cannot be used for archiving. In this case you would compress an archive file (like a tar file) to maximize usage.

11.9.6 zstd

[Zstandard](#) is a real-time compression algorithm, providing high compression ratios created at Facebook in 2015. It offers a very wide range of compression / speed trade-off, while being backed by a very fast decoder (see benchmarks below). It also offers a special mode for small data, called dictionary compression, and can create dictionaries from any sample set. Zstandard library is provided as open source software using a BSD license. It can be installed via the command: `sudo apt-get install zstd`.

11.9.7 tarballs

A tape archive that is additionally compressed by another tool is called a **tar ball** and the compression method is usually appended to the end of the filename.

Example Usage: `tar -cvzf code.tar.gz ./code-directory` This command will create a tar archive of the directory called code-directory and will compress it using the gzip compression algorithm by default. *Note the -z option added. Add a lowercase -j for bzip2 and uppercase -J for xz. Make sure to change the file extensions.

Example usage: Each one of these tar archives has been further compressed by one of the 4 Unix/Linux compression methods `file linux-4.3-rc3.tar.Z`; `file linux-4.3-rc3.tar.gzip`; `file linux-4.3-rc3.tar.bzip2`; `file linux-4.3-rc3.tar.xz`

Example usage: Previously you had to pass a flag to the tar command to tell it what type of compression algorithm to decompress with but now tar is smart and will autodetect for you. The flags you need to simply pass are -x for extract, -v for verbose (optional), and -f for file (optional) `wget`

²⁸<https://en.wikipedia.org/wiki/Compress>

²⁹<https://en.wikipedia.org/wiki/Gzip>

³⁰<https://en.wikipedia.org/wiki/Bzip2>

```
https://mirrors.edge.kernel.org/pub/linux/kernel/v5.x/linux-5.11.22.tar.xz; tar -xvJf
linux-5.11.22.tar.xz
```

As of version > 1.31 of `tar` there is also support for `zstd`. Which can be filter and archive via the `--zstd` flag.

11.10 Chapter Conclusions and Review

In this chapter we learned and mastered the concepts of disk based hardware. We learned about the types of Linux filesystems and the features of the new generation of filesystems. We learned how to install and configure volumes, snapshots, and compression. You have been prepared to with the basics of how to manage and understand the file-system on your Linux distro.

11.10.1 Review Questions

- 1) What is the `fdisk` program?
 - a) a dialog-driven program for the creation and manipulation of partition tables.
 - b) a filesystem creation tool
 - c) a tool for formatting floppy disks
 - d) a tool for removing disks from a system
- 2) What is the default VirtualBox disk type?
 - a) VDI
 - b) HDD
 - c) COW
 - d) VDD
- 3) After attaching a new virtual disk, what is the next step?
 - a) partitioning
 - b) make a filesystem
 - c) mount a filesystem
 - d) add an extent
- 4) Which command will print out currently all the block devices, their device name, and their partitions in a nice tree based format?
 - a) `lspci`
 - b) `lsblk`
 - c) `lsusb`
 - d) `lstr`
- 5) `fdisk` is part of what package?
 - a) `utils`
 - b) `GNU`
 - c) `utils-linux`
 - d) `utils-unix`
- 6) What would be the name of the second SATA disk attached to your system?
 - a) `sda`
 - b) `sdb`
 - c) `sdc`
 - d) `sdd`
- 7) What is the name of the first native Linux filesystem released in 1992?
 - a) `ext2`
 - b) `minix`
 - c) `ext4`
 - d) `ext`

- 8) What is the name of the current default Linux Filesystem?
- a) ext
 - b) btrfs
 - c) ext3
 - d) ext4
- 9) Ext4 breaks up data into _____, which is the smallest sized piece of data that can be read or written?
- a) sectors
 - b) tracks
 - c) blocks
 - d) clusters
- 10) If you use the ext2 filesystem and choose a 4 KiB block, what is the maximum filesystem size?
- a) 2 TiB
 - b) 16 TiB
 - c) 4 TiB
 - d) 4 GiB
- 11) What is the name of the maintainer of the ext4 filesystem?
- a) Brian Kernighan
 - b) Theodore Ts'o
 - c) Andrew Tanenbaum
 - d) Google
- 12) What is the name of the filesystem that the ext4 maintainer, Theodore Ts'o, is recommending to replace ext4?
- a) XFS
 - b) Btrfs
 - c) ZFS
 - d) HAMMER
- 13) What is the name of the filesystem that Red Hat adopted on their RHEL platform to replace ext4 and support better performance on large filesystems?
- a) ZFS
 - b) XFS
 - c) Btrfs
 - d) HAMMER
- 14) Which is the correct command needed to install on Ubuntu to be able to create XFS filesystems?
- a) `sudo apt-get install xfsprogs`
 - b) `sudo apt-get install zprogs`
 - c) `sudo apt-get install file-progs`
 - d) `sudo apt-get install zfsprogs`
- 15) What is the name of the combined filesystem and logical volume manager designed by Sun Microsystems?
- a) XFS
 - b) SunFS
 - c) ZFS
 - d) Btrfs
- 16) Which is the correct command for making an ext4 filesystem on a partition `/dev/sdb1`?
- a) `sudo mkfs.ext4 /dev/sdb`
 - b) `sudo mkfs.ext4 /dev/sdb1`
 - c) `sudo mkfs /dev/sdb1`
 - d) `sudo makefs`
- 17) Which is the correct command to mount an ext4 filesystem, `/dev/sdb1` on a mount point `/mnt/data-drive-2`?

- a) `sudo mnt /dev/sdb1 /mnt/data-drive-2`
- b) `sudo mnt -t ext4 /dev/sdb1 /mnt/data-drive-2`
- c) `sudo mount -t ext4 /dev/sdb1 /mnt/data-drive-2`
- d) `sudo mount /dev/sdb1 /mnt/data-drive-2`

18) Which file contains the mountpoints that will be mounted automatically at boot?

- a) `/etc/mntab`
- b) `/etc/default/grub.conf`
- c) `/etc/fstab`
- d) `/etc/tab`

19) What is the command used to create a LVM physical volume?

- a) `pvcreate`
- b) `pvck`
- c) `pvadd`
- d) `pvscan`

20) What is the command used to create a LVM volume group?

- a) `vgcreate`
- b) `vgck`
- c) `vgdisplay`
- d) `vgmknodes`

11.10.2 Podcast Questions

DragonFly BSD - Listen to this podcast: <https://ia802605.us.archive.org/9/items/bsdtalk248/bsdtalk248.mp3>

- ~1:25 What did DragonFly BSD drop with the 4.0 release?
- ~1:40 What was the other major feature that DragonFly BSD added?
- ~3:40 What modification did they add to the Packet Filter?
- ~10:00 What is the largest system DragonFly BSD has access to?
- ~11:45 What is the difference between DragonFly BSD's network stack compared to BSD and Linux?
- ~13:25 What is the limitations of the Hammer 1 Filesystem?
- ~13:45 What features will Hammer 2 Filesystem add?
- ~15:45 What is the intended use case of Hammer 2 FS?
- ~18:00 What sub-system is still in the works needed to make DragonFly BSD a stable work station?
- ~25:00 What is package-ng?
- ~30:00 How does DragonFly BSD handle suspend and resume functions common to laptops?
- ~35:50 What is the growing issue about systemd in relation to BSD?
- ~38:00 Of the 20,000 packages available in DragonFly BSD where are they primarily targeted?
- ~38:30 Out of FreeBSD, OpenBSD, NetBSD, and DragonFly – what is each project focusing on?
- ~40:23 How does GPL based Linux software cross over into BSD distros?

11.10.3 Lab

11.10.3.1 Lab 11 Objectives

- Creating virtual disks in VirtualBox
- Creating new partitions in fdisk
- Creating new filesystems with mkfs
- Creating new filesystems in ZFS and Btrfs
- Mounting new filesystems
- Editing `/etc/fstab` and using `systemd .mount` files to make our mounts permanent

11.10.3.2 Lab 11 Outcomes

At the conclusion of this lab you will have successfully created a new virtual disk in VirtualBox, created new partitions using fdisk, formatted those partitions using mkfs, XFS, and ZFS, and mounted all those partitions manually and automatically using the `/etc/fstab`.

11.10.3.3 Lab 11 Activities

1. Create 1 virtual drive in VirtualBox:
 - a. Use `fdisk` to create a primary partition
 - b. Format it with `ext4`
 - c. Mount it to `/mnt/disk1`
 - d. Add it to your `fstab`
2. Create 2 more virtual drives:
 - a. Create a single volume group named `vg-group`
 - b. Create 1 logical volume named `lv-group` using the two drives
 - c. Format it with `XFS`
 - d. Mount it to `/mnt/disk2`
 - e. Add the `lv-group` to your `fstab`, using the disk path to the logical volume not the UUID
 - f. Reboot the system and `cat` the `/etc/fstab` and show that your entry is present
3. Using the same LVM as before:
 - a. Add an additional VirtualBox disk and then create a LVM physical disk
 - b. Grow the volume group and logical volume
 - c. Grow the `XFS` file system
4. Using section 11.7.4 you will create a ZFS snapshot and roll back to it
 - a. Create a two disk ZFS stripe named `memorycache`
 - b. Change the ownership on the `/memorycache` volume to `controller:controller`
 - c. Change directory to `/memorycache` and display your `pwd`
 - d. Issue the command: `truncate -s 500m accounts.csv` to create a 500 mb file named `accounts.csv`
 - e. Create a ZFS snapshot of the `memorycache` volume named: `mc-snap1`
 - f. Using the `truncate` command create two more files: `ubunut-distros.csv` and `fedora-distros.csv` of 100 mb on the `/memorycache` volume
 - g. Issue the `ls -lh` command on the `/memorycache` volume to show that the new files have been created
 - h. Using the `zfs list` command list the current snapshots
 - i. Using the `zfs rollback` command the `mc-snap1` snapshot
 - j. Issue the `ls -lh` command on the `/memorycache` volume to show that the snapshot has been rolled back
5. Using Ubuntu 20.04 and ZFS, attach four 1 GB disks and create RAID 10 (a mirrored stripe). Name the pool: `datapool`. Display the `zpool status` and take a screenshot of the output.
6. Using Ubuntu 20.04, attach 4 virtual disks of 1 GB each. Create two Btrfs mirrored drives named `disk1` and `disk2`. Take a screenshot of the output of the `btrfs filesystem show` command for each disk.
7. Using Fedora, attach 4 1 GB disks in a Btrfs stripe.
 - a. Take a screenshot of the `btrfs filesystem df` command for this volume.
 - b. Then remove one of the virtual disks from the stripe. Take a screenshot of the `btrfs filesystem df` command for this volume.
 - c. Attach an additional 2 gb disk to the Btrfs stripe. Take a screenshot of the `btrfs filesystem df` command for this volume.
 - d. Extend the Btrfs filesystem to encompass using all of the new disk space. Take a screenshot of the `btrfs filesystem df` command for this volume.
8. On the `zpool` named `datapool` on Ubuntu 20.04 from question 5:
 - a. Execute a `zpool status` command
 - b. Enable LZ4 compression on the `zpool datapool`
 - c. Execute a `zfs get all | grep compression` command to display that compression is enabled
9. On your Fedora system execute, any of the commands listed to print out the disk serial numbers.

10. Attach an additional 2 GB virtual disk and format it with Btrfs and we will mount it in read-only mode. Using the command `lsblk --fs /dev/sdX` determine the UUID of the newest virtual disk you just created. Add an entry for this disk to the `/etc/fstab` file with the following values:
 - a. file system is UUID=
 - b. mount point is `/mnt/disk100` (create this partition if it doesn't exist)
 - c. type is `btrfs`
 - d. options: `defaults,ro` (ro for read-only)
 - e. dump and pass fields can be 0
 - f. Change owner and group to your username for `/mnt/disk100` (using `chmod`)
 - g. Reboot your system. Change directory to `/mnt/disk100` and take a screenshot to demonstrate that the disk is in read-only mode by trying to create a file via this command: `touch demo.txt`
11. Using `wget`, retrieve this URL: `https://mirrors.edge.kernel.org/pub/linux/kernel/v5.x/linux-5.11.19.tar.xz`
 - a. `Untar/uncompress` this archive.
 - b. `Tar` the directory and compress it using `bzip2`, make sure to keep the original input
 - c. `Tar` the directory and compress it using `gzip`, make sure to keep the original input
 - d. `Tar` the directory and compress it using `zstd`, make sure to keep the original input
 - e. `Tar` the directory and compress it using `xz`, make sure to keep the original input

11.10.3.4 Footnotes

Chapter 12

Networking, Webservers, and Intro to Databases

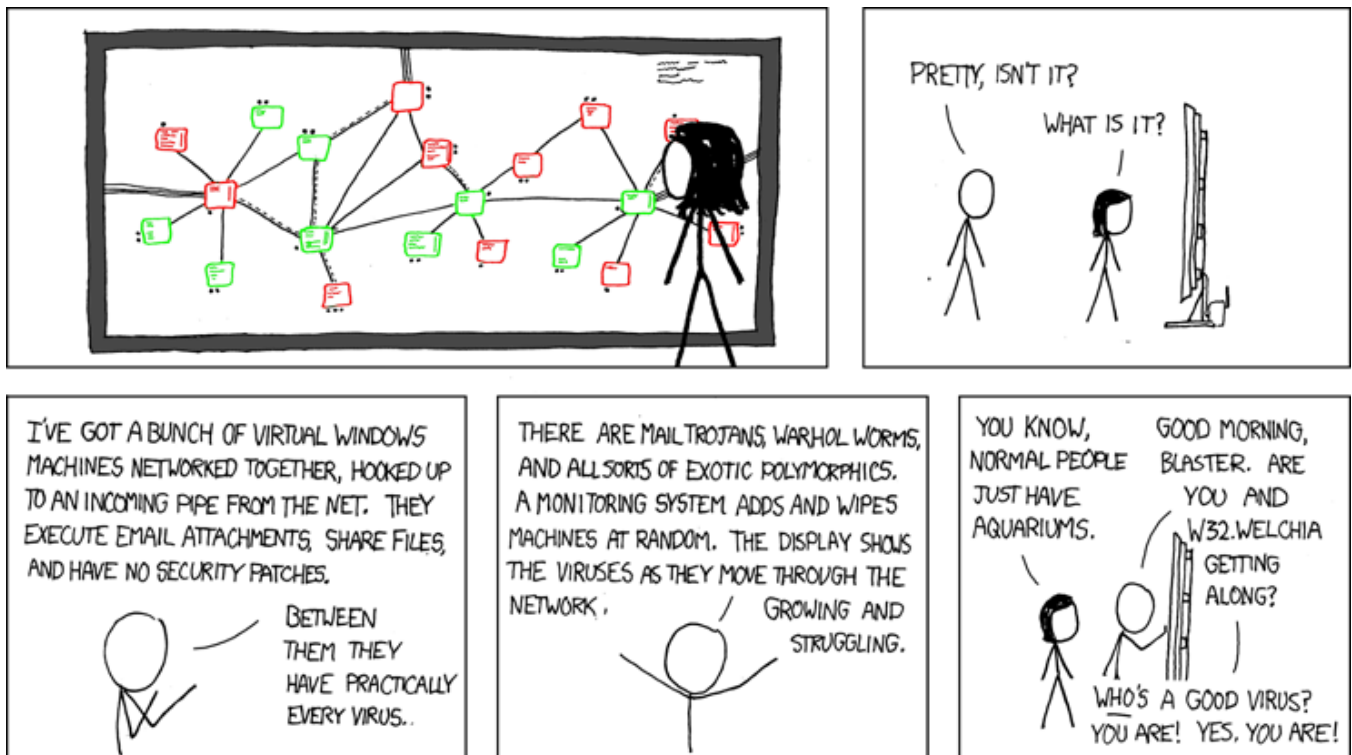


Figure 12.1: *Some people just have aquariums...*

12.1 Objectives

- Understand how to configure and display basic network settings for the two major Linux distribution families
- Understand how to use basic network troubleshooting tools
- Understand how to configure and deploy major webserver platforms for Linux and BSD
- Compare and contrast major open-source web servers and be able to discuss their uses in industry
- Understand how to install, manage, and configure basic major SQL and NOSQL databases

12.2 Outcomes

At the conclusion of this chapter you will have explored the nature of the network settings for the two major Linux distribution families and for BSD. You will be comfortable using various commandline troubleshooting tools to help solve network issues and problems. You will deploy two major webserver platforms and understand the differences between Apache and Nginx in their memory models and request handling models. Finally you will have gained experience deploying, installing, and configuring MySQL and MariaDB SQL databases and a popular NoSQL database MongoDB.

12.3 Networking

Former CEO of Sun Microsystems, Scott McNealy once famously said, “*The network is the computer.*” This was in 1990. He could not have been more correct. With this in mind, basic networking skills are mandatory. We will briefly cover topics in this order:

- IP addresses
 - Static and DHCP
 - MAC Address
- NETMASK and CIDR
- Gateway
- DNS

12.3.1 IP Addresses

Every network interface, or NIC, which is the physical or virtual place where your device connects to the network. Each NIC in turn needs an IP address to communicate on a network. IP addresses come in two flavors, a **static** and a **dynamic** IP address. An IPv4 or IP address is a 4 octet number looking something like this: 192.168.1.100.

Exercise: Open a command prompt and type this command to find your IP address: `ip address show`. This command can be abbreviated `ip a sh` as well.

A static address is one that you assign and configure based on your network. If you are the system administrator you can and should map each device on your network with its own IP address. For instance any servers you have, webservers, database servers, load balancers, routing equipment should have statically set IP information.

Exercise: Open a command prompt and type this command to find your MAC address or Ethernet Address `ip l sh` which is short for `ip link show`. You can find all the options to display by typing `man ip`.

But what if you have transient or ephemeral nodes (computers) on your network? Then you need to use the **Dynamic Host Configuration Protocol**. Setting your computer to use DHCP allows it to negotiate for a lease on a shared IP address. This is a good idea for transient devices or paces where the total number of IPs needed is less than the total number of devices, but all of those devices will not be present at the same time.

There is a DHCP server (configuring one is beyond the scope of this chapter), that will listen for DHCP broadcasts from your client and answer with an offer of an IP. Once your system (network card) accepts the offer it gains access to that IP address and all other necessary IP configuration—which is relinquishes upon your physically leaving the network for the most part. DHCP allows you to pool IPs when you might not have enough and share or allow for the auto-registration to make managing large scale IP deployments easy.

For instance at a university every student has a laptop and most likely a phone too, you could manually assign each an address but the number of students goes into the thousands and tens of thousands, and it not practical to manage—DHCP makes this scale manageable. Settings these values statically in each operating system is different but the concept is the same. You need to enter an IP Address, Netmask/CIDR, Network Gateway, and DNS. Each of these concepts is explained below.

12.3.2 MAC Address

Each network interface card or NIC has a 64 bit hardware address assigned to it. This is unique and split into two parts. The first three octets are the OUI, Organizational Unit Identifier, which is given to a particular company to help identify their products. The last three octets are random numbers that are chosen by the manufacturer after

the OUI is assigned for each device they manufacture. In some cases MAC addresses can be set via software. MAC addresses are used by switches to convert the last leg of a TCP/IP connection to an actual physical port and are at the second layer of the TCP/IP model.

```

enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq
link/ether 02:5f:45:77:d1:f5 brd ff:ff:ff:ff:ff:ff
inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s
    valid_lft 82852sec preferred_lft 82852sec
inet6 fe80::5f:45ff:fe77:d1f5/64 scope link
    valid_lft forever preferred_lft forever

```

Figure 12.2: Mac Addresses

12.3.2.1 Ubuntu

Canonical, that develops Ubuntu, keeps an excellent wiki with this information, [Ubuntu Network Wiki](#).

There are multiple ways to discover this information. There are two suites of tools. The original is `net-tools` the newer group is called the `iproute2` tools. If you have used a computer before, from BSD to Windows (which used the BSD TCP-stack) these commands will be familiar. But the *net-tools* suite development was actively **ceased** in 2001 in favor of [iproute2](#).

12.3.3 Comparison of Net-tools and iproute2

These look familiar don't they? The `ifconfig` command is a single command. To view other details such as the ARP table, RARP command, view or change routes you would have to use additional commands. As a contrast, the *iproute2* command handles all of that from the `ip` command. Older Linux (pre-2015) definitely have `net-tools` installed. That is quickly changing as some distributions are only including the `iproute2` package. One good example why to use the `iproute2` tools, is `net-tools` was created before IPv6 became a standard. There is a [iproute2 cheatsheet](#).

12.3.3.1 Replacement Commands Table

Table 12.1: [Commands and Their Replacements](#)

Net-Tools Deprecated Commands	<code>iproute2</code> Replacement Commands
<code>arp</code>	<code>ip n</code> (ip neighbor)
<code>ifconfig</code>	<code>ip a</code> (ip addr)
	<code>ip link</code>
<code>iptunnel</code>	<code>ip tunnel</code>
<code>iwconfig</code>	<code>iw</code>
<code>nameif</code>	<code>ip link</code> or <code>ifrename</code>
<code>netstat</code>	<code>ss</code>
	<code>ip route</code> (for <code>netstat -r</code>)
	<code>ip -s link</code> (for <code>netstat -i</code>)
	<code>ip maddr</code> (for <code>netstat -g</code>)
<code>route</code>	<code>ip r</code> (ip route)

12.3.4 udev and ethernet naming conventions under systemd

With the adoption of `systemd`, the convention for naming network cards changed from a driver based enumeration to [Predictable Network Interface Names](#).

*“Basic idea is that unlike previous *nix naming scheme where probing for hardware occurs in no particular order and may change between reboots, here interface name depends on physical location of hardware and can be predicted/guessed by looking at `lspci` or `lshw` output¹.”*

“The classic naming scheme for network interfaces applied by the kernel is to simply assign names beginning with “eth” to all interfaces as they are probed by the drivers. As the driver probing is generally not predictable for modern technology this means that as soon as multiple network interfaces are available the assignment of the names is generally not fixed anymore and it might very well happen that “eth0” on one boot ends up being “eth1” on the next. This can have serious security implications...²”

The systemd group [argued here](#):

The classic naming scheme for network interfaces applied by the kernel is to simply assign names beginning with “eth0”, “eth1”, ... to all interfaces as they are probed by the drivers. As the driver probing is generally not predictable for modern technology this means that as soon as multiple network interfaces are available the assignment of the names “eth0”, “eth1” and so on is generally not fixed anymore and it might very well happen that “eth0” on one boot ends up being “eth1” on the next. This can have serious security implications, for example in firewall rules which are coded for certain naming schemes, and which are hence very sensitive to unpredictable changing names.

The following different naming schemes for network interfaces are now supported by udev natively:

- 1) Names incorporating Firmware/BIOS provided index numbers for on-board devices (example: eno1)
- 2) Names incorporating Firmware/BIOS provided PCI Express hotplug slot index numbers (example: ens1)
- 3) Names incorporating physical/geographical location of the connector of the hardware (example: enp2s0)
- 4) Names incorporating the interfaces’s MAC address (example: enx78e7d1ea46da)
- 5) Classic, unpredictable kernel-native ethX naming (example: eth0)

What you gain by using this standard:

- Stable interface names across reboots
- Stable interface names even when hardware is added or removed
 - no re-enumeration takes place (to the level the firmware permits this)
- Stable interface names when kernels or drivers are updated/changed
- Stable interface names even if you have to replace broken ethernet cards by new ones

There is a short technical explanation of how these names are devised in the comments of the [source code here](#).

What does this mean? Well let us take a look at the output of the `ip a sh` command. Lets try it on Ubuntu 18.04, 16.04, Fedora 30, CentOS 7, and using `ifconfig` on FreeBSD 11 what do you see? On some of these you see eth0 some you see enp0sX. Why? Though all of these oses are using systemd, not FreeBSD, a few of them might have the value `biosdevname=0` set in their `/etc/default/grub` file, which we covered in chapter 10. The way to reset the values is listed below:

- Edit `/etc/default/grub`
- At the end of `GRUB_CMDLINE_LINUX` line append `net.ifnames=0 biosdevname=0`
- Save the file
- Type `grub2-mkconfig -o /boot/grub2/grub.cfg`
- or type `grub2-mkconfig -o /boot/efi/EFI/fedora/grub.cfg`
- reboot

[Source Source](#)

12.3.5 Network Configuration Troubles

Here is where things get tricky. In the future I would like to think this is will all be sorted out, but for now, buckle up. So networking was always controlled by a service under sysVinit, that was usually `sudo service networking restart`. This was common across all Linux. This worked fine when network connections were static and usually a 1 to 1 relationship with a computer or pc. That all changed as wireless connections became a reality, and the mobility of computers to move from network to network, and even virtual machines, that could be created and destroyed rapidly, all began to change how networking was done. In November of 2004 Fedora introduced **Network Manager**

¹<https://askubuntu.com/questions/704361/why-is-my-network-interface-named-enp0s25-instead-of-eth0?rq=1>

²<https://unix.stackexchange.com/questions/134483/why-is-my-ethernet-interface-called-enp0s10-instead-of-eth0>

to be the main instrument to handle their network configurations. Debian and Ubuntu would eventually follow behind and Network Manager became the default way to manage network connections. It uses a YAML like file structure to give values to the network service. Debian and Ubuntu maintained support for Network Manager, but always allowed fall back for compatibility reasons for the sysVinit script to manage the network.

The control of the network has been unified once again in all major Linux distros under **systemd-networkd**, which being part of systemd you assume that it controls the networking stack. Systemd-networkd will look for run time localized overwrites of default values located in `/etc/systemd/network`. Files in that directory need to end in a `.network` extension. The systemd-networkd `.network` file has an INI style value structure³: The entire systemd-networkd documentations is [described here](#).

12.3.5.1 Who uses what?

For the desktop Linux, Ubuntu and Fedora/Red Hat based, Network Manger is being used by default as of late 2021, but systemd-networkd can be enabled. The server edition of Ubuntu 20.04 use systemd-networkd.

12.3.5.2 Systemd-networkd network config file templates

```
# Systemd-networkd .network file (not Ubuntu Netplan)
# Name of the file /etc/systemd/network/20-wired.network
[Match]
Name=enp1s0

[Network]
DHCP=ipv4

# Wired adapter using a static IP
# /etc/systemd/network/20-wired.network
[Match]
Name=enp1s0

[Network]
Address=10.1.10.9/24
Gateway=10.1.10.1
DNS=10.1.10.1
#DNS=8.8.8.8
```

12.3.6 Ubuntu non-netplan Network Manager Config

This is the structure of the `/etc/network/interfaces` file Network Manager based file for Ubuntu and Debian (deprecated):

```
auto eth0
iface eth0 inet static
    address 192.168.0.42
    network 192.168.0.0
    netmask 255.255.255.0
    broadcast 192.168.0.255
    gateway 192.168.0.1

auto enp0s8
iface enp0s8 inet static
    address 192.168.0.42
    network 192.168.0.0
    netmask 255.255.255.0
    broadcast 192.168.0.255
    gateway 192.168.0.1
```

³[https://wiki/archlinux.org/index.php/Systemd-networkd](https://wiki.archlinux.org/index.php/Systemd-networkd)archlinux.org/index.php/Systemd-networkd “systemd-networkd”)

Here is the basic structure for the Network Manager based Fedora/CentOS systems located at⁴: `/etc/sysconfig/network-scripts` or `/etc/sysconfig/network-scripts/ifcfg-enp5s0`

```
DEVICE="eth0"
BOOTPROTO="dhcp"
ONBOOT="yes"
TYPE="Ethernet"
PERSISTENT_DHCLIENT="yes"

# Not all of these options are required
TYPE=Ethernet
BOOTPROTO=none
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
NAME=p3p1
UUID=7622e20e-3f2a-4b5c-83d8-f4f6e22ed7ec
ONBOOT=yes
DNS1=10.0.0.1
IPADDR0=10.0.0.2
PREFIX0=24
GATEWAY0=10.0.0.1
HWADDR=00:14:85:BC:1C:63
IPV6_PEERDNS=yes
IPV6_PEERROUTES=yes
```

12.3.6.1 Netplan.io

To further confuse things, Ubuntu decided to write a YAML based file management abstraction layer for networking that can use the same configuration for either Network Manager or Networkd-Systemd. It is called `netplan.io`. Netplan reads YAML style files from a network configuration located in `/etc/netplan/*.yaml`.

Not to be out done, the sample template from Netplan.io looks similar to `systemd-networkd`⁵. To configure `netplan`, save configuration files under `/etc/netplan/` with a `.yaml` extension (e.g. `/etc/netplan/config.yaml`), then run `sudo netplan apply`. This command parses and applies the configuration to the system. Configuration written to disk under `/etc/netplan/` will persist between reboots. By default in Ubuntu 18.04 Network Manager is used for actively managing network connections, Netplan is “on” but allows Network Manager to manage by default unless specifically altered below.

```
# To let the interface named 'enp3s0' get an address
# via DHCP, create a YAML file with the following:
network:
  version: 2
  renderer: networkd
  ethernets:
    enp3s0:
      dhcp4: true
```

To instead set a static IP address, use the `addresses` key, which takes a list of (IPv4 or IPv6), addresses along with the subnet prefix length (e.g. `/24`). Gateway and DNS information can be provided as well:

```
network:
  version: 2
  renderer: networkd
  ethernets:
```

⁴<https://superuser.com/questions/645487/static-ip-address-with-networkmanager>

⁵<https://netplan.io/examples>

```
enp3s0:
  addresses:
    - 10.10.10.2/24
  gateway4: 10.10.10.1
  nameservers:
    search: [mydomain, otherdomain]
    addresses: [10.10.10.1, 1.1.1.1]
```

12.3.6.2 Netmask

The netmask value or subnet of your network is actually a part of you IP address. So that routers know how to route packets to your network the netmask or network mask effectively blocks off a portion of your IP address. Traditionally netmasks were blocked into simple Class A, B, C, and D blocks, each one representing one of the IP octets. But this turned out to be highly inefficient. If you had a subnet of class A, your subnet would be 255.0.0.0. This means that you would be assigned a fixed value from 1-254 in your first IP octet and the remaining three octets would be variable. Apple famously has the 16.0.0.0 Class A giving them access to 255255255 IP addresses and Amazon recently received control of the 3.0.0.0 address block from GE.

Class B subnet is 255.255.0.0 and gives you access to 16,000 IP addresses ($254*254$) with the first two octets set. An example would be 172.24.x.y.

Class C address has 1 octet available and 3 octets preset. A common class C subnet you see mostly in home routing devices is 192.168.1.x which gives you 254 addresses. For our purposes we won't worry about class D and E in this book.

The problem is those division of IP octets are very clean, unfortunately leads to many wasted addresses. So a new way to divide blocks into smaller parts came along called [CIDR blocks](#) or blocking. CIDR lets you split blocks. A business might have a subnet or CIDR block of /23. This looks like a class B subnet. Class B in CIDR would be /24. The /23 gives us control of 192.168.1.110 and 192.168.1.111 for 512 addresses. /23 can be written as 255.255.254.0 as well.

12.3.6.3 Gateway

The gateway value is your networks default router. This value is literally the gateway in and out of your network. Usually this IP address ends in a .1 or a .254, but does not have to.

12.3.6.4 DNS

DNS—Domain Name services allow you to resolve written domain names. The sites google.com, web.iit.edu, twit.tv, etc, etc, turn those values via lookup into IP addresses that can then route packets to and from. DNS is very important. Without it you would have to remember the IP address of every single site your wanted to visit. Very quickly this wouldn't scale and in fact this idea of domain names lead to the initial founding of Yahoo as the personal index of its founder Jerry Wang in 1990s. DNS is now a native part of the internet and is maintained by core DNS servers that are scattered world wide. The predominant software being used for DNS is called BIND9 form the ISC, Internet Software Consortium. We will not configure DNS servers in this book, but focus on client configuration. Your ISP provides DNS for you, those come with some gray area of allowing ISPs to sell advertising on HTTP 404 error pages, or even inject advertising code into non-https based connections. There is a small list of alternative DNS services that give you free DNS in exchange for analyzing certain data in aggregate—beware before using them.

- Google has two public DNS services, [8.8.8.8](#) and [8.8.4.4](#)
- [Cloud Flare 1.1.1.1](#)
- [IBM Quad9 9.9.9.9](#)
- [OpenDNS servers](#)

DNS is set and configured as noted above in the various networking files. Note that DNS was not an initial part of TCP/IP networking so it was not natively contained in the network service configuration, DNS came later to the internet.

12.3.7 /etc/hosts

Linux, inheriting from UNIX from a time before DNS existed, has a file for local DNS lookups: `/etc/hosts`. This file is owned by root. You can edit this and place three items: an IP address, a fully qualified domain name, a short name

(just the hostname). This is enabled by default and is the first lookup for your system. This helps save network based DNS roundtrips and can be accessed by any application or script without needing modification or additional libraries.

```
# sample /etc/hosts file from a system setting up a sample network
```

```
192.168.33.110 riemanna.example.com riemanna
192.168.33.120 riemannb.example.com riemannb
192.168.33.100 riemannmc.example.com riemannmc

192.168.33.210 graphitea.example.com graphitea
192.168.33.220 graphiteb.example.com graphiteb
192.168.33.200 graphitemc.example.com graphitemc

192.168.33.10  hosta.example.com hosta
192.168.33.20  hostb.exampe.com hostb

192.168.33.50 logstash.example.com logstash
192.168.33.51 ela1.example.com ela1
192.168.33.52 ela2.example.com ela2
192.168.33.53 ela3.example.com ela3
```

12.3.7.1 iputils

Most of the time the network works fine, but when it doesn't you need to be able to use built in system tools to troubleshoot the problem and identify where the problem is. Those tools are separate from the iproute2 suite and are called [iputils](#). The tools included are listed here but all of them might not be installed by default.

- arping
- clockdiff
- ninfod
- ping
- rarpd
- rdisc
- tftpd
- tracepath
- traceroute
- traceroute6

The first tool that should be in your tool box is *ping*.

```
ping example ping www.google.com
ping 192.168.0.1
```

Ping, just like the concept of a submarine using sonar to find objects, communicates with another IP address to see if the other system is “alive” and that your system and network are working properly to get the packet from your network to a different network. There are many tools to enhance the output of ping as well such as [gping](#).

The `traceroute` tool is used to report each router hop that a packet takes on its way to its final destination. Useful for checking if there are routing problems along the path of your traffic. Try these commands and describe the output:

```
traceroute www.yahoo.com
traceroute www.yahoo.co.jp
```

There are additional tools that extend basic troubleshooting features such as:

- [iptraf-ng](#) - Network traffic visualization
- [tcpdump](#) - Detailed Network Traffic Analysis

12.4 Webservers

August 6th 1991, Tim Berners-Lee deployed the first webpage and the created the first webserver. For history's sake, an early copy of it was found on an old system and [restored](#). He was working at the [CERN](#) research lab in Switzerland. He did so with the idea to be able to freely share text data amongst researchers and national research labs world-wide. To do this he created the Hypertext Transfer Protocol – [HTTP](#) protocol for sending and receiving requests as well as a webserver named, NCSA, that would receive and process those requests, returning text to a client browser to be rendered.

The first webserver gave rise to a commercial company called Netscape started by the now famous investor Marc Andreesson, with research coming out of the University of Illinois. Famous for their Netscape Navigator browser, they were also the pioneers of the first webserver software. This software had been commercially available before at a high price and was limited to those who already could afford a large hardware investment. The Apache webserver came out of this code base and became the name and the first product of the Apache Opensource Foundation. When we look to two technologies that were used by the dotcom companies that survived the first and second crashes, we find that both the Apache webserver along with the MySQL database made the web possible.

Applications have grown enormously beyond just serving simple html webpages. What was once a simple webserver has been exchanged becoming application servers and serving traffic or aggregating traffic directly on a per request basis. We will cover in detail in the next chapters some of those application servers. For now let us start with webservers. They listen for requests on port 80. When receiving a request, they serve (they are webservers. . .) or render a page of HTML code and return that to a client (you) viewing a page through a web browser. The webserver by default will serve pages out of the `/var/www/html` directory on Linux and `/var/www` on FreeBSD.

12.4.1 Apache

Without Apache, companies such as Google, Facebook, Twitter, and many others started upon opensource never would have been able to get started.

Apache has over time grown and had to add new functions while shedding old functionality. The memory model of how it processes requests has changed over time as the frequency and amount of requests on a webserver has changed. Some may criticize Apache webserver for being a bit old, but there is a large body of knowledge out there on how to customize and manage it.

The Apache webserver can be installed via package managers. There is even a version of it available for Windows. Note that though the same application, Ubuntu refers to the Apache webserver as `apache2` and Red Hat products refer to it as `httpd`, which is not to be confused with the OpenBSD custom built webserver also named `httpd`.

```
sudo apt-get install apache2
sudo yum install httpd
```

```
#FreeBSD 12 using Ports
sudo portsnap fetch
sudo portsnap extract
sudo portsnap update
cd /usr/ports/www/apache2
make install
```

Webservers have various configurable components. The basic configuration out of the box is very conservative about resources and is not much use beyond for testing. You will need to tune the different settings as you go along as no two work loads are the same, unfortunately there is no direct tutorial you can just configure and run a large business with. For our purposes in class the default configurations will suffice, but in the real world you will need to find additional documents or books to guide you along.

Apache has extendable modules so its base features can be enhanced without needing to recompile the entire program. Using `apt-get` you can add modules that you can use to render the PHP language or modules to enable HTTP/2 capabilities for instance.

Let's try installing `apache2` and `php` at the same time and look at the dependency list:

- `sudo apt-get install apache2 php php-mysql mariadb-server`
- `sudo systemctl restart apache2` – (as opposed to `restart`) just re-reads the configurable

Sample code is shown here for a sample PHP webpage or copy and paste this code in to a file named: `index.php` located in `/var/www/html`

```
// Two slashes is a comment
<?php

echo phpinfo();

?>
```

You should be able to load this page in the browser inside your virtual machine by accessing: `http://localhost/index.php`

12.4.1.1 HTTP/2

“Websites that are efficient minimize the number of requests required to render an entire page by minifying (reducing the amount of code and packing smaller pieces of code into bundles, without reducing its ability to function) resources such as images and scripts. However, minification is not necessarily convenient nor efficient and may still require separate HTTP connections to get the page and the minified resources. HTTP/2 allows the server to “push” content, that is, to respond with data for more queries than the client requested. This allows the server to supply data it knows a web browser will need to render a web page, without waiting for the browser to examine the first response, and without the overhead of an additional request cycle.”

“Additional performance improvements in the first draft of HTTP/2 (which was a copy of SPDY) come from multiplexing of requests and responses to avoid the head-of-line blocking problem in HTTP 1 (even when HTTP pipelining is used), header compression, and prioritization of requests. HTTP/2 no longer supports HTTP 1.1’s chunked transfer encoding mechanism, as it provides its own, more efficient, mechanisms for data streaming⁶.”

The best resource I found was a technical deep-dive on HTTP/2 by [Steve Gibson at Security Now Podcast episode 495](#)

12.4.1.2 TLS Certs

One of the major innovations Netscape made with their original webserver product was the creation of SSL, secure socket layer technology. This allowed for sensitive data to be encrypted and decrypted securely—which enabled commerce over the internet to take off. HTTP connection using SSL have the prefix `https://`. SSL has long been deprecated and replaced with TLS - (Transport Layer Security) 1.2 and 1.3, but many people still use the phrase *SSL* when they really mean *TLS*.

You can configure your system to generate SSL certs, but they will be missing a key component of Certificates you can buy or receive from a third party. In that they don’t have a chain of trust about them. Self-signed certs will also trigger a browser to throw a security warning and block entry to that web-site. Now you have the option of overriding this and or accepting these self-signed certs into your operating systems certificate store. Some companies so this to secure internal traffic that does not go to the outside internet, but stays inside a company network.

There is an [EFF](#) led initiative called [Let’s Encrypt](#) that will give you free SSL certs for your public site. They offer wildcard domains and easy setup via `apt`, `yum`, and `dnf` to make this experience easy and remove all reasons to not encrypt web traffic. [You can see the adoption curve](#) of TLS/SSL since Let’s Encrypt became widely available.

- [TLS 1.3 Podcast on Security Now](#)
- [Lets Encrypt Explanation Podcast](#)
- [SSL Labs](#) is a free service that will check your TLS cert and server settings.
 - You can use [SSL labs](#) to check the Let’s Encrypt cert for [my own tech blog](#), [forge.sat.iit.edu](#).

Without having a public IP address you can’t use Let’s Encrypt, but you can generate a self-signed SSL/TLS certificate following these tutorials. Note that your browser will complain and send you dire warnings, you will have the option to accept the cert anyway and then the warnings will not persist.

- [Digital Ocean Nginx Self-Signed SSL Cert](#)
- [Digital Ocean Apache Self-Signed SSL Cert](#)

Just like anything you can, can automate the creation of a self-signed cert:

⁶<https://en.wikipedia.org/wiki/HTTP/2>


```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:4096 \  
-keyout /etc/ssl/private/apache-selfsigned.key \  
-out /etc/ssl/certs/apache-selfsigned.crt \  
-subj "/C=US/ST=Illinois/L=Chicago/O=IIT-Company/OU=Org/CN=www.school.com"
```

12.4.2 Nginx

Started in 2004 by Igor Sysoev, this product came out of a Russian company who found their unique web-serving needs couldn't be met by Apache. It is licensed under the [2 Clause BSD license](#). Apache had a memory model that was created when serving webpages in the the mid-1990s, and the nature of the web, including serving more dynamically generated pages, and information from multiple streams pushed Apache to the edge of its capability. Nginx was developed to overcome these limitations and solve the [C10K problem](#). Nginx has the ability to do load-balancing and reverse-proxying natively. Nginx achieves its speed increase by sacrificing the flexibility that Apache has.

CentOS and Fedora will need to add the `epel-release` package first, `sudo yum install epel-release` or `sudo dnf install epel-release`. For Ubuntu use `apt-get`.

12.4.3 OpenBSD httpd Process

The OpenBSD project which values security and home grown solutions over pure availability. Instead of trusting others code, the OpenBSD project built and maintain [their own webserver](#).

12.4.4 NodeJS

In late 2009/2010, a developer from [Joyent](#) (later Samsung/Joyent) wanted to explore the probabilities of JavaScript. Up to this time JavaScript had been used in the WebBrowser, but creator Ryan Dahl saw an opportunity. He took the [V8 JavaScript rendering engine](#) out of the Chrome browser, added an event loop and I/O functions and made it a standalone server. Now you could programmatically use JavaScript on the server-side as well as client-side called [Node.js](#). A package manager for Node was added a year later and called the Node Package manager or [NPM](#).

The Node.js release cycle is different then most major Linux distro's release cycles, so you need to go to the NodeJS site directly to get a newer version. For the latest 16.x LTS (long term support branch):

```
# Using Ubuntu  
curl -fsSL https://deb.nodesource.com/setup_16.x | sudo -E bash -  
sudo apt-get install -y nodejs  
node -v
```

```
# Using Debian, as root  
curl -fsSL https://deb.nodesource.com/setup_16.x | bash -  
apt-get install -y nodejs  
node -v
```

```
# Using Fedora/CentOS/Red Hat  
sudo dnf install -y gcc-c++ make  
curl -sL https://rpm.nodesource.com/setup_14.x | sudo -E bash -  
sudo dnf install nodejs  
node -v
```

Using the NPM package manager, we can install additional plugins that allow our Node.js JavaScript application to have additional features. For example:

```
# Using NPM to install the ExpressJS JavaScript server  
npm install express
```

```
# Using NPM to install the Mysql connector to talk to a MySQL or MariaDB database  
npm install mysql2
```

Now using vim or nano lets code a sample “Hello World” Node.js program, let’s create a directory in the *Documents* directory named: **myapp**. Name the file **app.js**. Then type the code sample below:

```
// Simple sample app from
// http://expressjs.com/en/starter/hello-world.html
// You can access this by opening a webbrowser on your Virtual Machine
// from the directory where the app.js is located run: node app.js
// and go to http://127.0.0.1:3000

const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`)
})
```

12.5 Database and NoSQL

Databases come in two types: **Relational databases** and **Non-relational databases (NoSQL)**. The relational database structure uses a query language called SQL, *Structured Query Language* which allows you to make queries on structured data. Structured data assumes that data is stored in typed fields such as integer, varchar, decimal, datetime, and so forth. These structured rows and columns are then stored in a table and accessed via the SQL syntax either via the command line or integrated into a programming language.

- SQL example - `SELECT answers FROM finalexam` or `SELECT * FROM EMPLOYEES WHERE ID=6000`
- NoSQL sample: `db.inventory.find({ status: { $in: ["A", "D"]} })`

12.5.1 MySQL and MariaDB

Installation of a database is straight forward using package managers, there are two pieces of the Relational Database (RDBMS) the client and the server. These parts do what they say, if you are accessing a database remotely, you do not need to install the entire server just the client tools to use the applications.

```
# Using Ubuntu or Debian based distros
# Install either mysql or mariadb
sudo apt-get install mariadb
sudo apt-get install mysql
```

```
sudo apt-get install mariadb-client
sudo apt-get install mariadb-server
```

```
# Installing on Fedora
sudo dnf install mariadb mariadb-server
sudo dnf install mariadb-client
# make sure to start and enable the maria or mysql service on Fedora/CentOS
```

MySQL was started by [Michael “Monte” Widens](#). The company was one of the first major companies to become successful with an opensource model, especially for a database product in a crowded market. MySQL the company was [sold to Sun in 2009](#), which then was inherited by Oracle in their purchase of Sun in 2010. Monte was not happy with Oracle’s stewardship of MySQL and decided to fork the codebase and begin a new yet familiar product called MariaDB. MariaDB continued the MySQL legacy by essentially restarting the MySQL company. MariaDB is for all purposes a drop in replacement for MySQL, even using the same commands to run the database. You can create a database and a table directly from the `mysql cli`)

- Log in
- Enter your password at the prompt
- Enter commands at the CLI
- Quit

```
-- On mysql/mariadb 8.x Fedora and Ubuntu
-- sudo mysql -u root
-- Create a database named: records
CREATE DATABASE records;

USE records;
create table tutorials_tbl(
    tutorial_id INT NOT NULL AUTO_INCREMENT,
    tutorial_title VARCHAR(100) NOT NULL,
    PRIMARY KEY ( tutorial_id )
);
-- Type quit to exit from the mysql cli
```

12.5.1.1 User Accounts and Security Concerns

After the installation of MariaDB/MySQL, you can create user accounts with limited privileges. This is a good idea as the `root` account doesn't have a password by default and is clearly recommended **NOT** to be used for anything other than administration. So your application will need to use a non-root account.

After you log in to the MySQL command-line using the command: `sudo mysql -u root`, and after you have created a database (as in the example above), you can create users and assign access to particular databases and even particular tables.

```
-- This is a comment
-- This statement grants only SELECT privileges, no edit privileges
-- creates a user named: worker
-- gives permission to all tables in the records database
-- from only the localhost IP address, 127.0.0.1
-- The basic permissions are: CREATE, SELECT, UPDATE, DELETE, and INSERT
GRANT SELECT ON records.* TO worker@'127.0.0.1' IDENTIFIED BY 'password-goes-here';
flush privileges;
```

You can place the previous SQL code that will create a table and enter a record into a single file. You can give it any name but convention says it should explain what the code does and end with `*.sql`. This code will be placed into a file named `create-table.sql` and the sample is located in the files `> chapter-12` directory.

```
CREATE DATABASE records;
USE records;
CREATE TABLE tutorials_tbl(
    tutorial_id INT NOT NULL AUTO_INCREMENT,
    tutorial_title VARCHAR(100) NOT NULL,
    PRIMARY KEY ( tutorial_id )
);

-- This code inserts a single record into the table for test purposes
INSERT INTO records(tutorial_title) VALUES('Best Book Ever');

# You can redirect input by having the create commands placed in a single file
sudo mysql -u root < ./create-table.sql
```

12.5.1.2 Inline SQL commands

SQL commands can also be executed inline as well.

```
sudo mysql -u root -e "CREATE DATABASE wordpress_db;"
sudo mysql -u root -e "CREATE USER 'wp_user'@'localhost' IDENTIFIED BY 'password';"
# You can assign long commands to variables and execute them too
```

```
COMMAND="GRANT ALL ON wordpress_db.* TO 'wp_user'@'localhost' IDENTIFIED BY 'password';"  
sudo mysql -u root -e "$COMMAND"  
sudo mysql -u root -e "FLUSH PRIVILEGES;"
```

12.5.2 PostgreSQL

As always in technology, product names often have a joke or a story behind them. PostgreSQL is no different. One of the original RDBMs, Ingress, was a product and a company in the 1980s. The successor to that project was PostgreSQL (see the pun?). PostgreSQL has the added advantage of being opensource, backed by a commercial company, as well as not being MySQL which is owned by Oracle. Installation is provided in custom repos that need to be added to a system before using a package manager.

- [PostgreSQL Downloads for Ubuntu and Fedora/CentOS](#)

12.5.3 SQLite

SQLite skips some of the bigger features to be mean and lean. “SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine⁷.” It is meant to store and retrieve data and that is about it. This makes it very small and very compact, which makes it great for using on the mobile platform Android or iOS since it is a single binary file, and can be installed on mobile devices and tablets as part of an application. Sqlite3 has the unique licensing of being the [Public Domain](#). You can install SQLite3 via the normal package mechanism and it is usually close to being up to date. Note that SQLite3 doesn’t listen on external ports by default it is included as an external library in your application.

- `sudo apt-get install sqlite3`
- `sudo yum install sqlite`
- `sudo dnf install sqlite`

12.5.4 MongoDB

Though there are many in this category, I have selected one NoSQL database. The difference here is that data is not stored in tables or typed fields but as simple untyped records—the NoSQL really refers to no relations or relational structure⁸. This means that records can be of any type or length. You access the data not through a Structured Query Language but using HTTP requests via REST; GET, PUT, PATCH and DELETE which mirror the functionality of CRUD—Create, Retrieve, Update, and Delete. This allows you to integrate your “query” language directly into your application code. REST is the outgrowth of the successful spread of HTTP as a protocol.

MongoDB packages are maintained by MongoDB – and are released outside of Linux distro release cycles. The installation process is different for Ubuntu and CentOS. The instructions to add a custom repository are located here:

- <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/>
- <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-red-hat/>
- Make sure to start the mongod service
- <https://docs.mongodb.com/manual/mongo/>
 - [Run Mongo insert sample](#)
 - [Run Mongo query sample](#)

12.5.5 Network File System - NFS

NFS, the network filesystem, is probably the most prominent network services using RPC. It allows to access files on remote hosts in exactly the same way as a user would access any local files. This is made possible by a mixture of kernel functionality on the client side (that uses the remote file system) and an NFS server on the server side (that provides the file data). This file access is completely transparent to the client, and works across a variety of server and host architectures. NFS offers a number of advantages⁹:

⁷<https://sqlite.org/about.html>

⁸<https://en.wikipedia.org/wiki/NoSQL>

⁹<https://tldp.org/LDP/nag/node140.html>

- Data accessed by all users can be kept on a central host, with clients mounting this directory at boot time. For example, you can keep all user accounts on one host, and have all hosts on your network mount /home from that host. If installed alongside with NIS, users can then log into any system, and still work on one set of files.
- Data consuming large amounts of disk space may be kept on a single host. For example, all files and programs relating to LaTeX and METAFONT could be kept and maintained in one place.
- Administrative data may be kept on a single host.

12.5.6 iSCSI

The [iSCSI protocol](#) is a reimplement of the SCSI disk communication protocol. SCSI was an alternative that could move data faster than the then ATA (pre-SATA) standard. Once SATA became available the SCSI based hardware was more expensive and was replaced by cheaper SATA and more standardized USB (for external devices). The SCSI bus was faster than the standardized ATA bus, but required a specialized adapter card and specialized cable to connect devices and external peripherals. Think of it pre-USB (circa 1998). This made SCSI desirable but expensive. Also the SCSI standard continued to improve throughput but at the cost of not being backwards compatible with older and other versions of SCSI, each had its own cables and connectors. By the year 2000 the SCSI protocol was well known and heavily invested in for server class hardware. In that year IBM and Cisco standardized the iSCSI protocol. iSCSI integrated SCSI commands to external targets over Ethernet/IP. Allowing you to separate your disks from storage and access them over a local network via the iSCSI protocol. Disks were formatted as LVMs or directly as a ZFS, Btrfs, or XFS based drives and then presented as **iSCSI targets** over the network. iSCSI has two components, the **iSCSI target** and the **iSCSI initiator**. The system that connects to a target in an **initiator**. iSCSI devices can replace the need for SAN technology (Storage Area Networks) and work on commodity hardware over basic ethernet cables and switches.

This allows you to separate your storage and your compute. You can even use iSCSI disks as your main hard drive and configure this during install time on most major Linux distros. All modern Operating Systems come with support for being either a target or an initiator. A company called [iXsystems](#) has made a business out of providing ZFS based iSCSI storage devices running FreeBSD.

12.6 Firewall

Used to block external communication on you system ports. Not unlike plugs in the wall of your home, your server has ports that different services connect to and communication on. This allows the operating system and applications to communicate as well with multiple programs. There are 65000 ports available to use. The first 1024 ports are reserved for well known services. These numbers are useful to know, but applications have changed. For instance SMTP is no longer unencrypted and used over port 25, but port 587 or 995. Also the use of Git over http has replaced the need for FTP/SFTP/SSH and other protocols to send and retrieve data.

- SSH - 22
- FTP - 21
- SMTP - 25 (deprecated not used as it is an unsecured transport method)
- DNS - 53
- HTTP - 80 (becoming deprecated in browsers)
- HTTPS - 443 (HTTP with TLS/SSL)
- SMTP over SSL - 990
- MongoDB - 27017
- PostgreSQL - 5432
- MySQL - 3306
- Oracle DB - 1521
- ExpressJS - 3000

You can use rules to allow or deny traffic based on source IP, source Port, Destination IP, or Destination Port. Some people urge turning the firewall off because of complexity. I do not recommend this. If you are going to run a business, you need to understand what ports are open and why—opening them all is not a solution and could be a violation of laws regarding security, privacy, and government regulation.

12.6.1 Firewalld

Distributions using systemd have switched to [firewalld](#) as their main firewall interface. There had been previous ways to interface with a firewall and firewalld seeks to abstract these away and present a unified interface to your systems. Fedora turns their firewall on by default, CentOS 7 does not.

Firewalld uses the `firewall-cmd` command and not `firewallctl` like you would expect. It has a concept of *zones* which allow you to predefine a collection of rules that can be applied to different zones. Permanent configuration is loaded from XML files in `/usr/lib/firewalld` or `/etc/firewalld`. When adding a new rule you need to declare if the rule is permanent or it will be reset when the firewalld service is reloaded. The firewalld system contains zones such as:

- trusted or untrusted
- drop
 - incoming packets are dropped, outbound packets are allowed
- block
 - incoming packets are rejected with an icmp-host-prohibited response
- public
- work
- home
- internal

```
# Firewalld Documentation and Examples
# https://firewalld.org/
# https://firewalld.org/documentation/
```

```
# Install firewalld on Ubuntu
sudo apt-get install firewalld
sudo systemctl enable firewalld
sudo systemctl start firewalld
```

```
# Commands to check the state/status of firewalls
sudo systemctl status firewalld
sudo firewall-cmd --state
sudo firewall-cmd --get-active-zones
```

```
# Firewalld any OS, how to add exceptions for services or ports
sudo firewall-cmd --zone=public --add-service=http --permanent
sudo firewall-cmd --zone=public --add-port=22/tcp --permanent
sudo firewall-cmd --zone=public --list-ports
# Needed to reload the process changes to the firewall
sudo firewall-cmd --reload
```

```
# specific IPs -- changes the semantics
# https://serverfault.com/questions/680780/block-all-but-a-few-ips-with-firewalld
```

```
firewall-cmd --zone=public --add-service=ssh
firewall-cmd --zone=public --add-source=192.168.56.105/32
firewall-cmd --zone=public --add-source=192.168.56.120/32
firewall-cmd --zone=public --remove-service=ssh
# Needed to reload the changed to the firewall
sudo firewall-cmd --reload
```

12.6.1.1 fail2ban

The main reason to have a firewall is to restrict traffic to your system or server. Note the commands above do not dictate in anyway who can connect to a system. Firewalld includes a standard interface so third party tools and build integration into your firewall. [Fail2ban](#) is an anti-bruteforce tool for systems that have their connections exposed to the public network, such as MySQL and openssh-server. It allows you do ban IP addresses that are trying to brute force hack your system. You can do permanent banning or a timeout based banning. [Fail2ban](#) has a firewalld integration where you can add firewall rules to block bad IPs automatically.

```
# you may need the epel-release package on Fedora/CentOS
# https://fedoraproject.org/wiki/Fail2ban_with_Firewalld
# https://unix.stackexchange.com/questions/268357/
```

```
sudo dnf install fail2ban fail2ban-firewalld
sudo apt-get install fail2ban fail2ban-firewalld
sudo yum install fail2ban fail2ban-firewalld
```

```
sudo systemctl enable fail2ban
sudo systemctl start fail2ban
```

12.6.2 Ubuntu UFW

Ubuntu uses [UFW \(Uncomplicated Firewall\)](#). It is installed but not enabled by default. You can enable it: `sudo ufw enable` and print out the status via `sudo ufw status`. Syntax examples include:

- Allow or deny by port number
 - `sudo ufw allow 53`
 - `sudo ufw deny 53`
- Allow or deny by service
 - `sudo ufw allow ssh`
 - `sudo ufw deny ssh`
- Allow from IP
 - `sudo ufw allow from 207.46.232.182`
- Allow from subnet/CIDR block
 - `sudo ufw allow from 192.168.1.0/24`
 - `sudo ufw allow from 192.168.0.4 to any port 22`
 - `sudo ufw allow from 192.168.0.4 to any port 22 proto tcp`
- Enable/Disable ufw logging
 - `sudo ufw logging on`
 - `sudo ufw logging off`

Firewalld can be installed on Ubuntu via `apt-get` and then enabled and started as a service in place of UFW if you want to maintain that service and not use UFW.

12.7 Chapter Conclusions and Review

In this chapter we learned about the basic components of networking. We learned how to configure these settings and general network troubleshooting which will allow you to fully administer any system you come in contact with.

12.7.1 Review Questions

- 1) Using the ip2 suite of tools, which command(s) would show your IP address?
 - a. `ifconfig`
 - b. `ipconfig`
 - c. `ip address show`
 - d. `ip a sh`
- 2) Using the ip2 suite of tools, which command would show your routing table?
 - a. `ss`
 - b. `route`
 - c. `ip route show`
 - d. `ip -r`
- 3) What tool could you use to establish if a server is responding to requests?
 - a. `pong`
 - b. `ping`

- c. google
 - d. traceroute
- 4) What is the purpose of a netmask?
- 5) What is the purpose of DNS?
- 6) What is the name of the systemd firewall?
- a. systemd-firewalld
 - b. systemd-firewall
 - c. firewalld-cmd
 - d. ufw
- 7) What would be the command to list all of the firewalld public zone ports in use?
- a. `sudo systemctl status firewalld`
 - b. `sudo firewall-cmd --zone=public --list-all`
 - c. `sudo firewall-cmd --zone=public --list-all`
 - d. `sudo firewall-cmd --list-all`
- 8) If you had a CIDR block of /24 and a network address of 192.168.1.0, how many IP addresses would you have?
- a. 10
 - b. 0
 - c. 24
 - d. 256
- 9) What is the default port for HTTPS (TLS/SSL)?
- a. 80
 - b. 3000
 - c. 8080
 - d. 443
- 10) Using Network Manager, what tool is used to release a DHCP address from the command line?
- a. `rhclient`
 - b. `ipconfig /release`
 - c. `dhclient -r`
 - d. `xclient -r`
- 11) Using the ip2 suite, what command can be used to monitor and examine all current local ports and TCP/IP connections?
- a. `ss`
 - b. `net`
 - c. `wireshark`
 - d. `netstat`
- 12) Where are your network card settings located on Ubuntu while using Network Manager?
- 13) Where are your network card settings located on Fedora using Network Manager?
- 14) Where are your network card settings located on Ubuntu using Netplan?
- 15) What are the two major opensource webservers?
- a. Apache, Nginx
 - b. openhttpd, Nginx
 - c. Apache, IIS
 - d. Apache, Tomcat
- 16) What are two related and major opensource relational databases?
- a. SQL and MySQL
 - b. MariaDB and MySQL

- c. MySQL and Oracle DB
 - d. Nginx and MySQL
- 17) What command would you type to get to the MySQL command line prompt as the root user?
- 18) What is the file location that the system uses as a *local DNS* for resolving IP?
- a. `etc/systemd/hosts`
 - b. `/etc/hosts`
 - c. `/etc/allow`
 - d. `/etc/etc/etc`
- 19) What flag would you add to this command to make it survive a reboot: `sudo firewall-cmd --zone=public --add-port=22/tcp`
- a. `--peppermint`
 - b. `--permenant`
 - c. `--allow`
 - d. `--list-all`
- 20) Before systemd, NIC interface naming schemes depended on a driver based enumeration process. They switched to a predictable network interface naming process that depends on what for the interface names?
- a. driver loading order
 - b. interface names depend on physical location of hardware (bus enumeration)
 - c. kernel version
 - d. What ever Lennart Poettering feels like naming them

12.7.2 Podcast Questions

View or listen to this Podcast about Nginx: <http://twit.tv/show/floss-weekly/283>

- 1) ~2:02 What is Nginx?
- 2) ~3:22 What percentage of the world's websites are served with Nginx (as of 2014)?
- 3) ~4:57 What was the challenge that led to the creation of Nginx?
- 4) ~5:33 What is the main architectural difference between Nginx and Apache web servers?
- 5) ~8:32 What are some of the main use cases for Nginx?
- 6) ~11:00 When did Sarah get involved in Nginx?
- 7) ~16:23 How do you pronounce "Nginx"?
- 8) ~17:41 What is "caching" in relation to websites?
- 9) ~19:45 What is "proxying" in relation to websites?
- 10) ~29:36 What was the founder's motive to open source Nginx?
- 11) ~34:00 What is the difference in the open source Nginx and the commercial version? (Freemium?)
- 12) ~40:19 Are there Linux distro packages for Nginx?
- 13) ~53:10 Can Apache and Nginx co-exist or is it a winner take all?

12.7.3 Lab

12.7.3.1 Pre-reqs and Assumptions

Using two virtual machines, while powered off, in the VirtualBox settings, enable a second bridged network interface and set the type to **Bridged Adapter** (details are in last chapter and the VirtualBox networking details are in chapter 03).

- 1) Use the command to identify the IP address of each of the two systems
 - a. Capture a screenshot of both system's IP addresses
 - b. Use the `ping` tool to ping the each others IP and its results (`ctrl +C` to quit), take a screenshot of the results
 - c. Modify the `/etc/hosts` file and add an entry for both system in both systems give them the hostname `host1` and `host2`
 - d. Execute the `ping` command again this time using the hostname declared in the `/etc/hosts` file and capture a screenshot of the results

- 2) Use the command to display you MAC address of the network connection used in question 1, and in your screenshot highlight the MAC address
- 3) Use the command to display your IP address of the network connection used in question 1, and in your screenshot highlight the Netmask/CIDR block
- 4) On Ubuntu Desktop, Fedora Desktop, and Ubuntu Server – determine if you are running Systemd-Networkd or Network-Manager. On each system run these commands and take a screenshot of each results:
 - a. `sudo systemctl status systemd-networkd`
 - b. `sudo systemctl status NetworkManager`
- 5) On Ubuntu and Fedora Desktop, use the command: `sudo systemctl status firewalld` check to see if firewalld is enabled, if its not installed, use the package manager to install the package `firewalld`
- 6) Using firewalld, open port 22 permanently to allow SSH connections to your Fedora system, take a screenshot of the command `sudo firewalld-cmd --zone=public --list-all` to show the port is open
- 7) Using firewalld, open port 80 permanently to allow SSH connections to your Fedora system, take a screenshot of the command `sudo firewalld-cmd --zone=public --list-all` to show the port is open
- 8) If needed, install Nginx Webserver, and enable the proper firewall port (443) to serve pages over **https** following [this Digital Ocean configuration tutorial to create a self-signed cert](#). Take a screenshot of the webbrowser showing the <https://127.0.0.1>
- 9) On an OS of your choice, install Node.js version 16.x and use NPM to install the **express** package. Using the sample, “Hello World” code provided in the chapter, take a screenshot of the output of opening a browser on your Virtual Machine at the URL: `http://127.0.0.1:3000` – **Note** - from the directory where your `app.js` file is you will need to run `node app.js` to start the server and make sure that port 3000 is open in the firewall
- 10) Going to WordPress.org and download the latest tar.gz file. Follow the 5 minute setup to configure a working WordPress blog – installing all the needed dependencies, initialize the WordPress system and create a simple blog post named: “Hello World” as the screenshot to prove the work was done on the OS of your choice

12.7.3.2 Footnotes

Chapter 13

Infrastructure Installation and IT Orchestration

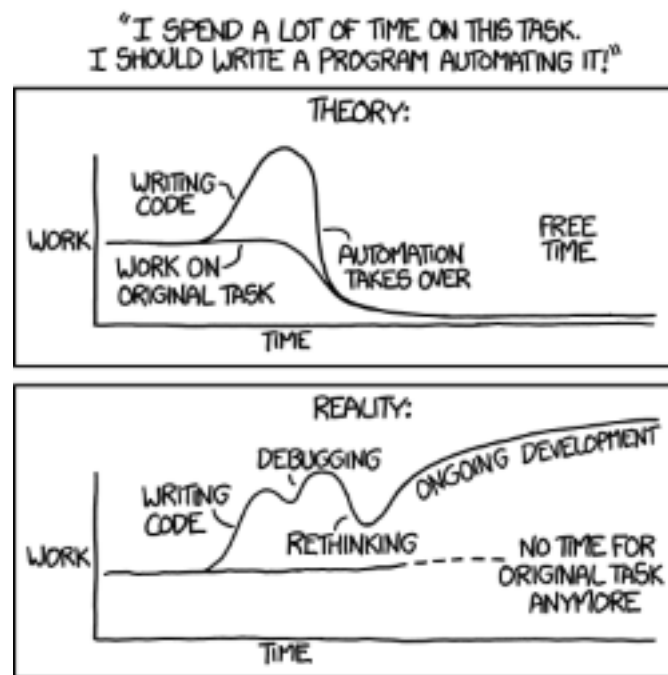


Figure 13.1: *Good thing we avoided that one...*

13.1 Objectives

- Linux Orchestration and Automation tools
 - Vagrant
 - Packer
 - Automation tools
 - Secrets Management

13.2 Outcomes

At the conclusion of this chapter you will have a basic understanding of how to use infrastructure automation and orchestration tools. You will be familiar and able to explain the concept of immutable infrastructure and will be able

to use Linux commands for enabling cloud native development technologies.

13.3 Automation and HashiCorp

One of the main things that computers are good at is executing repetitive tasks. One of the things humans seem to dislike is repeating the same task. Let's give a practical example; there is a divide between the developers (devs) and the operations (ops) people when it comes to software and infrastructure (dev + ops = devops). Developers need hardware to test their code on and then more hardware to run their code in production. Operations people have to maintain those systems and the code and the lifecycle of the application. Developers started to think, could we deploy our infrastructure in the same way we deploy code? Could we automate the way out of this problem? A young developer named Mitchell Hashimoto had the same thoughts.



Mitchell realized that himself and a good majority of developers were using VirtualBox to run and build test systems on their local machines and laptops. He found that he could be productive by keeping copies of the production servers where his code would reside as a local vm on his system. He realized there were some limitations to what VirtualBox would allow for in the way of access to the system. VirtualBox just provided an interface to virtualize an operating system, and had no provisions for quickly deploying or automating an OS deployment—it was still a manual process. He and his company set out to solve this problem by developing opensource software. They developed this stack of software:

You can learn more about HashiCorp at this [HashiConf 2017 keynote presentation](#) from Mitchell Hashimoto.

13.4 Vagrant

13.4.1 The Problem

The problem is that we have high levels of computing available to us yet we are not really using any of it. Our own laptops are fully powered and used as primary workstations. Yet when it comes to developers writing code, QA testers, Operations and Security people, these resources are not being used—we have ad-hoc efforts to gain some hardware, any hardware, for testing. This leads to a fractured development and inconsistent experiences, which often lead to short cuts and compromises that you are forced to pay later.

Enter Mitchell Hashimoto and HashiCorp. They started in 2010 with a radical mission to automate everything they could into repeatable steps using single tools that handle abstractions. HashiCorp was platform agnostic and focused on the process of automation. They succeeded and are now considered the industry leaders in the arena. Quite frankly there are no other tools that even compete in this space. <https://www.hashicorp.com/files/DevOps-Defined.pdf>.

At that time, [HashiCorp](#) was born. This was in 2010 and Mitchell's first task was easing the deployment, connection, and most importantly abstracting the network address translation between host and guest operating systems. He created [Vagrant](#) to do just this. Vagrant initially was a VirtualBox only product but has moved to be an abstraction layer now for multiple virtualization and container platforms

¹http://d13pix9kaak6wt.cloudfront.net/background/users/m/i/t/mitchellh_1370739801_5.jpg

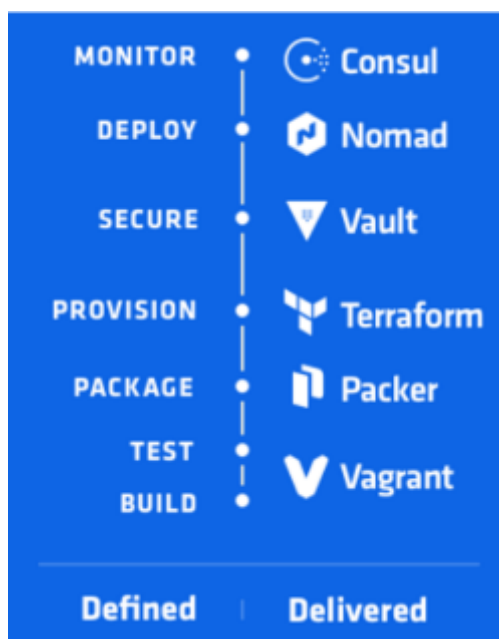


Figure 13.2: HashiCorp Stack

13.4.2 How Vagrant Benefits You²

If you are a developer, Vagrant will isolate dependencies and their configuration within a single disposable, consistent environment, without sacrificing any of the tools you are used to working with (editors, browsers, debuggers, etc.). Once you or someone else creates a single Vagrantfile, you just need to vagrant up and everything is installed and configured for you to work. Other members of your team create their development environments from the same configuration, so whether you are working on Linux, Mac OS X, or Windows, all your team members are running code in the same environment, against the same dependencies, all configured the same way. Say goodbye to “works on my machine” bugs.

If you are an operations engineer, Vagrant gives you a disposable environment and consistent workflow for developing and testing infrastructure management scripts. You can quickly test things like shell scripts, Chef cookbooks, Puppet modules, and more using local virtualization such as VirtualBox or VMware. Then, with the same configuration, you can test these scripts on remote clouds such as AWS or RackSpace with the same workflow. Ditch your custom scripts to recycle EC2 instances, stop juggling SSH prompts to various machines, and start using Vagrant to bring sanity to your life.

If you are a designer, Vagrant will automatically set everything up that is required for that web app in order for you to focus on doing what you do best: design. Once a developer configures Vagrant, you do not need to worry about how to get that app running ever again. No more bothering other developers to help you fix your environment so you can test designs. Just check out the code, vagrant up, and start designing.

Think of Vagrant as an abstraction layer between you and VirtualBox, Hyper-V, Docker, or even VMware desktop. It is written in the Ruby Language and comes as a self-contained binary that runs across all platforms. For the duration of this chapter I will use VirtualBox as my example. Vagrant handles this abstraction by using a file concept called a **box** or ***.box**. The box file is nothing more than a compressed archive containing a virtual hard drive and a configuration file that tells the Vagrant provider which virtualization software to launch this with. For example a ***.box** file that was made for the VirtualBox provider would contain a ***.vmdk** (hard drive) and an ***.ovf** file (meta-data and Virtual Machine settings file). Each Vagrant ***.box** file needs a config file called: **Vagrantfile**. This is an abstraction file to modify settings for the virtual machine at run time. There is a sample Vagrantfile later in this chapter. These two components are what is needed to run and manage Vagrant boxes.

²<https://www.vagrantup.com/docs/why-vagrant/>

13.4.2.1 Obtaining a Vagrant Box

There are two ways to obtain a Vagrant Box (*.box file). The first way would be to obtain pre-made images from a site you trust (remember you are running other people's configuration and software in your place of work – just be aware). The first place to look is from Vagrantup.com itself - <https://app.vagrantup.com/boxes/search>. Here you can search for boxes of other operating systems and versions even some opensource companies release a pre-configured Vagrant Box all setup for you to test their software all in one place. Using this facility you can simply run a command from the command line to add this box to your local system. such as: `vagrant init ubuntu/bionic64` would automatically construct a `Vagrantfile`, as well as retrieve an Ubuntu Bionic64 box file. The second way is described later in the chapter; that is to build and add your own box.

13.4.2.2 Vagrant Box Commands

When executing the `vagrant box` command from the command line (in Windows recommend using PowerShell Core) you will see this list of subcommands below, but we will primarily use just the first three:

Table 13.1: `vagrant box` commands

```
-----  
  add  
  list  
  remove  
  outdated  
  repackage  
  update  
-----
```

13.4.2.3 `vagrant box add`

```
vagrant box add
```

The first command `add` is the command we will use to add boxes (either of the two methods) from above. These are all premade systems made with Packer.io and [distributed by HashiCorp](#).

The tutorial on vagrantup.com will walk you through this but a small example (try any of these especially if you are not familiar with these platforms).

- `vagrant box add centos/7` (Official CentOS 7 Vagrant box release)
- `vagrant box add centos/8` (Official CentOS 8 Vagrant box release)
- `vagrant box add debian/buster64` (Debian provided release of Debian 10 x64)
- `vagrant box add terrywang/archlinux` (user provided Arch Linux distro)
- `vagrant box add laravel/homestead` (Preconfigured PHP Laravel framework development box)
- `Vagrant box add generic/Fedora32` (Fedora 32 server edition)
- `vagrant box add freebsd/FreeBSD-12.0-CURRENT` (official FreeBSD vagrant box)
- `vagrant box add maier/alpine-3.4-x86_64` (user provided alpine Linux distro)
- `vagrant box add ubuntu/bionic64` (Canonical-Ubuntu 18.04 parent company - provided)

You may need to use a full URL in the case of downloading a Vagrant box that is not provided from HashiCorp box repositories. This goes for 3rd party and for the boxes you create on your own. We will learn how to make our own in the Packer.io section of this document, but for all purposes the artifacts are the same; a *.box file. For installing a [Devuan](#) box (the distro that resulted from the Debian Civil War/systemd split) here are two ways to execute the commands:

```
# These are Linux/Mac based line continuations '\ ' otherwise this  
# code should all go on one line  
vagrant box add \  
http://devuan.ksx4system.net/devuan_beowulf/minimal-live/  
devuan_beowulf_3.0.0_amd64_minimal-live.iso \  
--name devuan-beowulf
```

```
vagrant box add ./itmd521-virtualbox-1503930054.box --name itmd-521
```

Adding a box via URL both ways, requires an additional parameter, `--name` (as seen above). The `--name` option is something you declare for your use, just don't put any spaces and its best to name the box something related to the actual box; `"box1"` or `"thebox"` are terrible names.

13.4.2.4 vagrant box list

`vagrant box list`

You can check to see if the vagrant box add command was successful by issuing the command: `vagrant box list`; looking something like this: (Note this is my system, yours will vary but the structure will be the same).

```
PS C:\Users\Jeremy\Documents\vagrant> vagrant box list
centos-vanilla-1908 (virtualbox, 0)
ubuntu-vanilla-18045 (virtualbox, 0)
ubuntu/bionic64 (virtualbox, 20200324.0.0)
ubuntu/xenial64 (virtualbox, 20200326.0.0)
```

Here you notice that the last two boxes were added directly from the HashiCorp boxes repository (`vagrant box add ubuntu/bionic64` and `vagrant box add ubuntu/xenial64`)

The top two boxes were custom Vagrant boxes I created (we are getting to that part) that are treated as third party boxes. To add them I issued a command like this: (The vanilla term is my own convention, it just means this is a default OS install – no extra packages)

```
vagrant box add ./centos-1908-virtualbox-1485312680.box --name centos-7-vanilla
vagrant box add ./ubuntu-18045-virtualbox-1598457730.box --name ubuntu-18045-vanilla
```

13.4.2.5 vagrant box remove

`vagrant box remove`

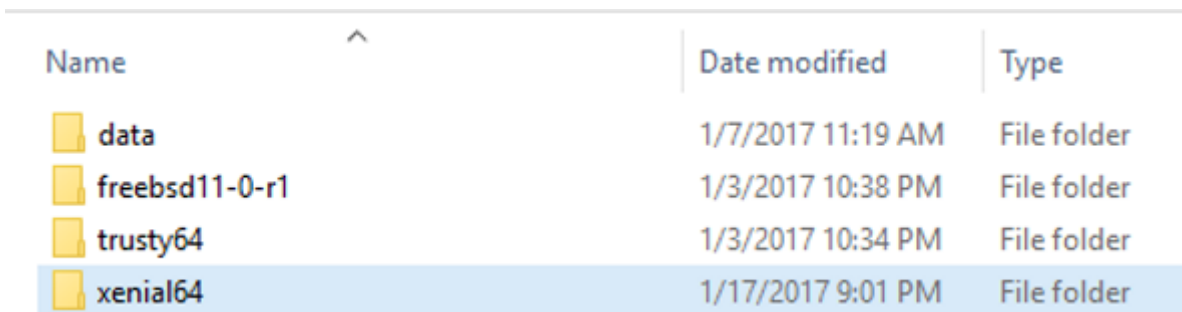
The same way that you add boxes you can remove them from your list. You need to know the name of the box that was added, run a `vagrant box list` command and find the name that way. The below commands would remove the boxes added in the previous section.

- `vagrant box remove centos-7-vanilla`
- `vagrant box remove ubuntu-18045-vanilla`

13.4.2.6 vagrant init

`vagrant init`

Once your Vagrant boxes have been added to your system and Vagrant has them in a list, you can now create a Vagrantfile. You have one Vagrantfile per-Vagrant box. It would make the most sense to create a sub-directory with the same box name to house the Vagrantfile. Then it would make sense to create a folder named after each of these boxes under the directory `vagrant`. In this example I created the directory `vagrant` under `Documents`. Why? It seems to make logical sense.



Name	Date modified	Type
data	1/7/2017 11:19 AM	File folder
freebsd11-0-r1	1/3/2017 10:38 PM	File folder
trusty64	1/3/2017 10:34 PM	File folder
xenial64	1/17/2017 9:01 PM	File folder

Figure 13.3: *Suggested Folder Hierarchy*

Note—for good measure I added a directory called `data` which will be used for mounting shared drives. Once you have created these folders, `cd` into one. For instance take the `trusty64` and `xenial64`. You would `cd` into `trusty64` directory and type: `vagrant init ubuntu/bionic64`. This will create a file called `Vagrantfile` that points and works with the `trusty64` vagrant box. The idea behind the `Vagrantfile` is that it has a shorthand syntax that is universally translated by Vagrant into specific virtualization platforms. The `Vagrantfile` handles all the properties that could be set (such as RAM, CPU, shared drives, port forwarding, networking, and so forth). Make sure you issue the `vagrant init` command from inside of the proper folder you just created.

Here is a sample `Vagrantfile`, which is available in the book source code [files > Chapter-13 > vagrant-init-files](#):

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# All Vagrant configuration is done below. The "2" in Vagrant.configure
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know what
# you're doing.
Vagrant.configure("2") do |config|
  # The most common configuration options are documented and commented below.
  # For a complete reference, please see the online documentation at
  # https://docs.vagrantup.com.

  # Every Vagrant development environment requires a box. You can search for
  # boxes at https://app.vagrantup.com/boxes/search.
  config.vm.box = "ubuntu/bionic64"

  # Disable automatic box update checking. If you disable this, then
  # boxes will only be checked for updates when the user runs
  # `vagrant box outdated`. This is not recommended.
  config.vm.box_check_update = false

  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine. In the example below,
  # accessing "localhost:8080" will access port 80 on the guest machine.
  config.vm.network "forwarded_port", guest: 8080, host: 8080

  # Create a private network, which allows host-only access to the machine
  # using a specific IP.
  config.vm.network "private_network", ip: "192.168.33.10"

  # Create a public network, which generally matched to bridged network.
  # Bridged networks make the machine appear as another physical device on
  # your network.
  config.vm.network "public_network"

  # Share an additional folder to the guest VM. The first argument is
  # the path on the host to the actual folder. The second argument is
  # the path on the guest to mount the folder. And the optional third
  # argument is a set of non-required options.
  config.vm.synced_folder "./data", "/vagrant_data"

  # Provider-specific configuration so you can fine-tune various
  # backing providers for Vagrant. These expose provider-specific options.
  # Example for VirtualBox:
  #
  config.vm.provider "virtualbox" do |vb|
    # Display the VirtualBox GUI when booting the machine
```



```

#   vb.gui = true
#
#   # Customize the amount of memory on the VM:
vb.memory = "2048"
end
#
# View the documentation for the provider you are using for more
# information on available options.

# Define a Vagrant Push strategy for pushing to Atlas. Other push strategies
# such as FTP and Heroku are also available. See the documentation at
# https://docs.vagrantup.com/v2/push/atlas.html for more information.
# config.push.define "atlas" do |push|
#   push.app = "YOUR_ATLAS_USERNAME/YOUR_APPLICATION_NAME"
# end

# Enable provisioning with a shell script. Additional provisioners such as
# Puppet, Chef, Ansible, Salt, and Docker are also available. Please see the
# documentation for more information about their specific syntax and use.
# config.vm.provision "shell", inline: <<-SHELL
#   apt-get update
#   apt-get install -y apache2
# SHELL
end

```

13.4.2.7 vagrant up

Command: `vagrant up`

Once your Vagrantfile has been created the next step to launch the virtual machine via Vagrant is through the `vagrant up` command. You would issue the command from the same directory where the Vagrantfile is located. A `vagrant up` command looks in the local directory for a Vagrantfile to begin parsing. This command is akin to starting the virtual machine directly. On the first run the Vagrantfile will be parsed and any settings in the virtual machine platform (VirtualBox in our case) will be changed. On subsequent runs the Vagrantfile will be ignored. **Note** - This command is issued not from inside the virtual machine but from the commandline of the host system.

Command: `vagrant up --provision`

The `--provision` flag tells Vagrant to re-provision and re-read and parse the Vagrantfile and make any additional changes while launching the virtual machine. Note - This command is issued not from inside the virtual machine but from the commandline of the host system.

Command: `vagrant up --provider virtualbox | hyperv | docker | vmware`

When using a Vagrant box from HashiCorp or any other it is a good idea to use the `-provider` flag to tell Vagrant which platform it will be virtualizing. This is optional but if you experience problems this is a good troubleshooting tip.

13.4.2.8 vagrant reload

Command: `vagrant reload`

This is akin to a reboot or restart of a virtual machine. Note - This command is issued not from inside the virtual machine but from the commandline of the host system.

Command: `vagrant reload --provision`

Will restart the system as well as re-read and parse the Vagrantfile. Note - This command is issued not from inside the virtual machine but from the commandline of the host system.

13.4.2.9 vagrant suspend

Command: `vagrant suspend`

This will put the virtual machine in suspend or pause mode (standby) as opposed to running `vagrant halt`, which will power the virtual machine off. Very handy to quickly resume work. Don't expect the system to automatically put your virtual machine into standby if you are used to just closing the lid of your laptop. Note - This command is issued not from inside the virtual machine but from the commandline of the host system.

13.4.2.10 vagrant halt

Command: `vagrant halt` Full shutdown of the virtual machine (power off). Note - This command is issued not from inside the virtual machine but from the commandline of the host system.

13.4.2.11 vagrant destroy

Command: `vagrant destroy`

This command is used to destroy the current instance of a virtual machine – but not remove the source files. This allows you to issue a `vagrant up` command and “start from scratch” without rebuilding or reinstalling the Vagrant Box. Note - This command is issued not from inside the virtual machine but from the commandline of the host system.

13.4.2.12 vagrant ssh

Command: `vagrant ssh`

This command is issued after the `vagrant up` command and allows you to establish an SSH session directly into the vagrant box, with a pre-setup username and password, with NO ASK set in the sudoers file, making for seamless entry. You should never need to access a username or password in Vagrant as that defeats the purpose of Vagrant. But for completeness's sake it is `vagrant:vagrant`. NOTE - the `vagrant ssh` command works perfectly by default on all Linux, macOS, and Windows 10 hosts.

13.4.2.13 vagrant plugin install

Command: `vagrant plugin install vagrant-vbguest`

This command specifically enables the automatic installation of the VirtualBox Additions to enable VirtualBox specific features such as shared folders.

13.4.3 Vagrant Quick Command Tutorial

Here is a small walk through to install 2 different Vagrant boxes:

1. Create a directory called `vagrant` on your host system (not in a virtual machine)
2. In that directory create 2 sub-directories; `bionic64` and `centos7`
3. `cd` to the `bionic64` directory and issue this command: `vagrant init ubuntu/bionic64`
4. Issue the command `vagrant up`
5. Upon successful boot, issue the command: `vagrant ssh` to connect to `bionic64` virtual machine - then exit the ssh session
6. Repeat the above steps in the `centos7` directory and replace the `init` command in step 3 with: `vagrant init centos/7`
7. In each directory issue the command `vagrant halt` or `vagrant suspend` to power down the VMs

13.5 Packer

13.5.1 The Problem Packer Solves

By 2010 Vagrant was being used to manage VMs, there was no tool that could be used to quickly and reliably create VMs. This problem was solved by HashiCorp and called [Packer](#). Packer, much like the name suggests, allows you to automate the installation of operating systems. or better said, “Packer is a tool for creating machine and container

images for multiple platforms from a single source configuration³.” Operating systems from Windows to Linux to BSD were all designed to be installed manually. Unlike installing software, there is no existing operating system when you are installing an operating system, making automatic installation difficult.

Packer attacked this problem by creating its own binary which acts as a supervisor and initiates the proper key sequence to turn a manual install into and automated install via a JSON based build template. This can take place on multiple formats or platforms and does not even focus on physical machines. Packer uses already existing answer file technology for Linux, such as Kickstart and Preseed to allow for automated and repeatable installs to create machine images. “A machine image is a single static unit that contains a pre-configured operating system and installed software which is used to quickly create new running machines. Machine image formats change for each platform. Some examples include AMIs for EC2, VMDK/VMX files for VMware, OVF exports for VirtualBox, and others⁴.” Network installs have existed for decades, but this method always assumed a physical 1-to-1 infrastructure, which as you have seen in this class is no longer the only reality.

Having a code based automated configuration now lets you track, audit, and centralize your build template in version control. With minor modifications, you can now have centralized machine image construction and export to various hardware platforms. You could build a VirtualBox VM, which could be exported to Vagrant or Amazon Web Services, or Docker. Now all of your developers, operations, testers, and QA can have access to the same machine on most any platform. As stated on the [Packer.io](https://www.packer.io) webpage the advantages of using packer are as follows⁵:

- **Super fast infrastructure deployment**
 - Packer images allow you to launch completely provisioned and configured machines in seconds rather than several minutes or hours. This benefits not only production, but development as well, since development virtual machines can also be launched in seconds, without waiting for a typically much longer provisioning time.
- **Multi-provider portability**
 - Because Packer creates identical images for multiple platforms, you can run production in AWS, staging/QA in a private cloud like OpenStack, and development in desktop virtualization solutions such as VMware or VirtualBox. Each environment is running an identical machine image, giving ultimate portability.
- **Improved stability**
 - Packer installs and configures all the software for a machine at the time the image is built. If there are bugs in these scripts, they’ll be caught early, rather than several minutes after a machine is launched.
- **Greater testability**
 - After a machine image is built, that machine image can be quickly launched and smoke tested to verify that things appear to be working. If they are, you can be confident that any other machines launched from that image will function properly.

Packer examples and example build code <https://github.com/jhajek/packer-vagrant-build-scripts.git>

13.5.1.1 Packer - Conventions

HashiCorp essentially built a tool that captures each install step. These steps are placed into a Packer build template or just template for short. These templates are constructed using **JSON**. In addition these templates rely on an “Answer File” for completing all of the installation choices and automating the installation. On Linux this “answer file” is split between the major Linux distribution families:

13.5.1.2 Packer JSON Build Template

Let us look at an example JSON template file: This source can be retrieved from the source code of the book: [files > Chapter-13 > packer-build-templates > ubuntu18045-vanilla.json](#)

```
{
  "builders": [
    {
      "name": "ubuntu-vanilla-18045-server",
      "vm_name": "ubuntu-vanilla-18045-server",
      "type": "virtualbox-iso",
```

³<https://www.packer.io/intro/index.html>

⁴<https://www.packer.io/intro/index.html>

⁵<https://www.packer.io/intro/why.html>

```

"boot_command": [
  "<esc><wait>",
  "<esc><wait>",
  "<enter><wait>",
  "/install/vmlinuz<wait>",
  " auto<wait>",
  " console-setup/ask_detect=false<wait>",
  " console-setup/layoutcode=us<wait>",
  " console-setup/modelcode=pc105<wait>",
  " debconf/frontend=noninteractive<wait>",
  " debian-installer=en_US<wait>",
  " fb=false<wait>",
  " initrd=/install/initrd.gz<wait>",
  " kbd-chooser/method=us<wait>",
  " keyboard-configuration/layout=USA<wait>",
  " keyboard-configuration/variant=USA<wait>",
  " locale=en_US<wait>",
  " netcfg/get_domain=vm<wait>",
  " netcfg/get_hostname=vagrant<wait>",
  " grub-installer/bootdev=/dev/sda<wait>",
  " noapic<wait>",
  " preseed/url=http://{{ .HTTPIP }}:{{ .HTTPPort }}/preseed/preseed.cfg<wait>",
  " -- <wait>",
  "<enter><wait>"
],
"boot_wait": "10s",
"disk_size": 20000,
"guest_os_type": "Ubuntu_64",
"http_directory": ".",
"http_port_min": 9001,
"http_port_max": 9001,
"iso_urls": "http://cdimage.ubuntu.com/ubuntu/releases/bionic/release/ubuntu-18.04.5-server-amd64.iso",
"iso_checksum": "sha256:8c5fc24894394035402f66f3824beb7234b757dd2b5531379cb310cedfdf0996",
"ssh_username": "vagrant",
"ssh_password": "vagrant",
"ssh_port": 22,
"ssh_wait_timeout": "10000s",
"shutdown_command": "echo 'vagrant'|sudo -S shutdown -P now",
"guest_additions_mode": "disable",
"guest_additions_path": "VBoxGuestAdditions_{{.Version}}.iso",
"virtualbox_version_file": ".vbox_version",
"vboxmanage": [
  [
    "modifyvm",
    "{{.Name}}",
    "--memory",
    "2048"
  ]
]
}
],
"provisioners": [
  {
    "type": "shell",
    "execute_command": "echo 'vagrant' | {{.Vars}} sudo -E -S sh '{{.Path}}'",
    "script": "../scripts/post_install_vagrant.sh"
  }
]

```

```

],
"post-processors": [
  {
    "type": "vagrant",
    "keep_input_artifact": false,
    "output": "../build/{{.BuildName}}-{{.Provider}}-{{timestamp}}.box"
  }
]
}

```

There are 3 sections we are interested in:

1. builders
2. provisioners
3. post-processors

13.5.1.3 Builders

The majority of this information is taken from <https://www.packer.io/docs/>. Builders are the initial syntax needed to build for a single platform. This forms the bulk of the JSON Key-Value pairs you see above in the sample template I provided. The Builder of choice above is for VirtualBox initially, but if my target platform had been something else then I could have switched. Note the syntax will be different for each builder as some require things others do not (Amazon and Azure require account keys for instance). The builder in the template above is a VirtualBox ISO builder. That is defined on line 2: "type": "virtualbox-iso". Many of these values will change or be different depending on the builder you use. Consult the documentation.

The builders available are:

1. Builders
2. Alicloud ECS
3. Amazon EC2
4. Azure
5. CloudStack
6. DigitalOcean
7. Docker
8. File
9. Google Cloud
10. Hetzner Cloud
11. HyperOne
12. Hyper-V
13. Linode
14. LXC
15. LXD
16. NAVER Cloud
17. Null
18. 1&1
19. OpenStack
20. Oracle
21. Outscale
22. Parallels
23. ProfitBricks
24. QEMU
25. Scaleway
26. Tencent Cloud
27. JDCloud
28. Triton
29. UCloud
30. Vagrant
31. VirtualBox

- 32. VMware
- 33. Yandex.Cloud
- 34. Custom

13.5.1.4 How the Operating System gets installed

In the builder, there is an `iso_url` and `iso_checksum` values that will retrieve installation media and run a checksum against it to make sure that the file is not damaged or corrupt.

```
"iso_urls": ["http://cdimage.ubuntu.com/ubuntu/releases/bionic/release/ubuntu-18.04.5-server-amd64.iso"],
"iso_checksum": "8c5fc24894394035402f66f3824beb7234b757dd2b5531379cb310cedfdf0996",
```

This URL is the actual remote location of the install media which will be retrieved and cached into a directory named `packer_cache`. The iso file will be renamed with the `iso_checksum`. Subsequent `packer_build` commands will use this cached iso media. You can also copy this cached iso file to other directories as long as it is placed in a local `packer_cache` directory—this can be used to speed up downloads.

13.5.1.5 Provisioners

Provisioner are tools that you can use to customize your machine image after the base install is finished. Though tempting to just use the Kickstart or Pressed files to do the custom install—this is not a good idea. You should leave the “answer files” as clean or basic as possible so that you may reuse them and do your customization here via a provisioner.

```
"provisioners": [
  {
    "type": "shell",
    "execute_command" : "echo 'vagrant' | {{ .Vars }} sudo -E -S sh '{{ .Path }}'",
    "script": "../scripts/post_install_vagrant.sh"
  }
]
```

In the sample above I chose to implement an inline shell command, “execute command” and then via a shell script. Shell scripts are very easy to use and flexible. Provisioners can also be connected to use Provisioning 3rd party tools such as Puppet, Chef, Salt, Ansible, as well as PowerShell. These tools are called Orchestration tools and I would recommend checking them out if your interest or job lies in this domain.

If you are using Packer to build Vagrant boxes, this code below is needed to be included in a shell script that is run in the provisioners block via a script key-value pair. Contents would look like below, and you would add any additional code after these lines. This is nothing but a shell script so any Linux commands you would normally execute in a shell script can be executed here.

```
#!/bin/bash
set -e
set -v

# http://superuser.com/questions/196848/how-do-i-create-an-administrator-user-on-ubuntu
# http://unix.stackexchange.com/questions/1416/redirecting-stdout-to-a-file-you-dont-have-write-permission
# This line assumes the user you created in the preseed directory is vagrant
echo "vagrant ALL=(ALL) NOPASSWD:ALL" | sudo tee /etc/sudoers.d/init-users
sudo cat /etc/sudoers.d/init-users

# Installing vagrant keys
wget --no-check-certificate 'https://raw.githubusercontent.com/mitchellh/vagrant/master/keys/vagrant.pub'
sudo mkdir -p /home/vagrant/.ssh
sudo chown -R vagrant:vagrant /home/vagrant/.ssh
cat ./vagrant.pub >> /home/vagrant/.ssh/authorized_keys
sudo chown -R vagrant:vagrant /home/vagrant/.ssh/authorized_keys
echo "All Done!"
```

```
# Add additional code here
```

Provisioners allow you to have multiple provision scripts. Some people like to split functionality over multiple shell scripts or use multiple methods. The `shell` provisioner also has the capability to execute inline Linux commands, in case you need to setup some internal structure. Also you can use this method to copy code into the Virtual Machine you are created. If you clone a Git repo to your local system, you can copy that application code directly into your virtual machine as Packer is building it.

```
{
  "type": "shell",
  "execute_command": "echo 'vagrant' | {{ .Vars }} sudo -E -S sh '{{ .Path }}'",
  "inline": [
    "mkdir -p /home/vagrant/.ssh",
    "mkdir -p /root/.ssh",
    "chmod 600 /home/vagrant/id_rsa_github_deploy_key",
    "cp -v /home/vagrant/id_rsa_github_deploy_key /home/vagrant/.ssh/",
    "cp -v ./applicaiton-code/html /var/www/html/",
    "cp -v ./mysql/.my.cnf /home/vagrant",
  ]
}
```

13.5.1.6 Post-Processors

Packer has the ability to build a virtual machine or OS Container once and export it to many different types of platforms in a single execution stretch. The initial artifact can be exported and converted across all of the formats listed below. Therein lies the power of Packer as you can deploy your production environment to any platform for any person: Dev, QA, Test, Ops, Sec, and so forth.

Once the Build step and Provision step are complete the last step (which is optional) is the post-processor step. This is where you can convert your base image you built into various other formats. Note that the directory named “build” is completely arbitrary and was created by me as it made sense.

```
"post-processors": [
  {
    "type": "vagrant",
    "keep_input_artifact": true,
    "output": "../build/{{.BuildName}}-{{.Provider}}-{{timestamp}}.box"
  }
]
```

Here you see I am converting the VirtualBox `.ovf` file into a Vagrant Box file `.box`. If you leave off the `keep_input_artifact` option, the initial artifact will be deleted and only the post-processor result will remain. If you are concerned about hard drive space - set this value to false.

Post-Processing includes:

1. Alicloud Import
2. Amazon Import
3. Artifice
4. Compress
5. Checksum
6. DigitalOcean Import
7. Docker Import
8. Docker Push
9. Docker Save
10. Docker Tag
11. Exoscale Import
12. Google Compute Export
13. Google Compute Import
14. Manifest

15. Shell (Local)
16. Vagrant
17. Vagrant Cloud
18. vSphere
19. vSphere Template

You can use multiple post-processors if desired.

13.5.1.7 vboxmanage

This command allows you to issue custom VirtualBox commands from within Packer. This is helpful for modifying hardware, such as number of CPUs and memory allocated during the install. Also you can modify the number of attached disks programmatically.

```
"vboxmanage": [
  [
    "modifyvm",
    "{{.Name}}",
    "--memory",
    "2048"
  ]
]

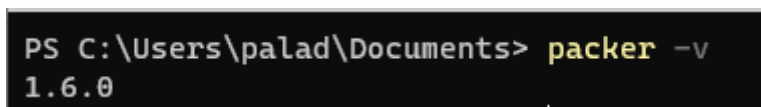
"vboxmanage": [
  [
    "modifyvm",
    "{{.Name}}",
    "--memory",
    "2048"
  ],
  ["createhd", "--filename", "output-virtualbox/packer-virtualbox-disk3.vdi", "--size", "15000", "--format", "vdi"],
  ["storageattach", "{{.Name}}", "--storagectl", "IDE Controller", "--port", "1", "--device", "0", "--type", "hdd"],
  ["createhd", "--filename", "output-virtualbox/packer-virtualbox-disk4.vdi", "--size", "15000", "--format", "vdi"],
  ["storageattach", "{{.Name}}", "--storagectl", "IDE Controller", "--port", "1", "--device", "0", "--type", "hdd"]
]
```

13.5.1.8 Packer, Putting it Together

There are two key commands to use in Packer, assuming you have added the packer executable to your system path. First type: `packer -v` and see if you get version information output.

Command: `packer -v`

Execute this command to see if you get a version output. If this command throws an error – go back and check the PATH & INSTALL section and make sure you have closed and reopened your shell.



```
PS C:\Users\palad\Documents> packer -v
1.6.0
```

Figure 13.4: Output of `packer -v`

Command: `packer validate`

This command will check the syntax of your *.json packer template for syntax errors or missing brackets. It will not check logic but just syntax. Good idea to run it to make sure everything is in order. Using the samples provided in the GitHub repo you can validate the *.JSON template with this command:

Command: `packer build`

This command will be what is used to execute and run the packer *.json template.


```
PS C:\Users\controller\Documents> packer validate .\ubuntu18041-vanilla.json
Template validated successfully.
```

Figure 13.5: *Output of packer validate*

```
PS D:\vanilla-install> packer build .\ubuntu18041-vanilla.json
ubuntu-vanilla-18041-server output will be in this color.

==> ubuntu-vanilla-18041-server: Retrieving ISO
ubuntu-vanilla-18041-server: Found already downloaded, initial checksum matched,
ubuntu.com/ubuntu/releases/bionic/release/ubuntu-18.04-server-amd64.iso
==> ubuntu-vanilla-18041-server: Starting HTTP server on port 9001
==> ubuntu-vanilla-18041-server: Creating virtual machine...
==> ubuntu-vanilla-18041-server: Creating hard drive...
```

Figure 13.6: *Output of packer build*

Packer Environment Variables: PACKER_CACHE_DIR

Data (U:) > packer_cache





<input type="checkbox"/> Name	Date modified	Type	Size
 7fed57084c8648a485d61f6dd091efc133d02ff8d60a65dcb0eff...	November 27	Disc Image File	927,744 KB
 1afad4c786ec4ddc4ef93048f5489c48656da25e663d603aa73ea...	September 1	Disc Image File	890,880 KB
 1b6d82288e807df0e4e080d740a92904f2197e9dbbfff065d10c2...	October 31	Disc Image File	874,044 KB
 ef29b8dd390aff66db69a3e1676a512af65f95f5249d8f21c01b34...	March 12	Disc Image File	868,352 KB

Figure 13.7: *Packer Cache Directory on Windows*

When running `packer build` Packer will cache the the install media—the iso. You can set this location to a central directory as this will save time downloading the same media over and over. On Windows you can configure the `PACKER_CACHE_DIR` by setting a file location in your user account environment variables. In Linux and Mac you can set an environment variable in your user profile.

13.5.1.9 When Packer Fails

If any part of the Packer build step fails or generates an error, Packer will roll back and delete any artifacts it has created. It won't leave you with broken items. Sometimes Packer fails right on the last step without much of an error message. In this case I recommend just re-run the packer build command. Upon completion, cd into the build directory that was created during the post-processor process and go back to the beginning of the Vagrant Tutorial and the section regarding using vagrant box add of custom boxes.

13.5.2 Answer Files

Most operating system installations are designed for a manual install process. This makes sense as for almost 40 years we have been using PC's (Personal Computers). A first step in automating the install process is to automate the answering of the installation questions. This is the most repetitive process of the install, as well as something that is not conducive to the human, as you spend most of your time waiting.

13.5.2.1 Fedora and Kickstart

The first solution came from Sun in 1994 and was called [Jumpstart](#). This was used to hold installation information and as a target system was booted, it would communicate to the Jumpstart server and complete the entire install over the network, OS and handle configuration.

The next phase came in Linux with Fedora creating the [Kickstart](#) answer file system. Kickstart does not handle the part of OS install, but the OS configuration and software retrieval/installation. Kickstart files can be generated from a

template or scratch or upon a successful install a default kickstart is located in the location `/root/anaconda-ks.cfg`. Examples and explanation resources can be found here:

- [Fedora Kickstart Reference](#)
- [CentOS Kickstart Reference](#)

13.5.2.2 Debian, Ubuntu, and Preseed

[Debian/Ubuntu pressed template](#) Debian created their own answer file system called [preseed]. You can interrupt a manual install and point to a kickstart file, but it needs to be done over the network. When you are installing an operating system you don't yet have a filesystem to read files from!

Working example of a preseed and a kickstart file can be found in the source code of the book: `files > Chapter-13 > packer-build-templates > preseed` and `files > Chapter-13 > packer-build-templates > ks`

Preseed Used for all Debian and Ubuntu based server installs - example and explanation resources can be found here:

- [Sample Preseed Template](#)
- [Preseed Ubuntu Guide](#)

13.5.3 Putting Vagrant and Packer together

How then do we build our own artifacts with Packer to manage them? Here is an end-to-end example using some sample code provided in the source code repo of the book. This example will use a prepared Packer build template to install and configure a Vanilla version of Ubuntu Server 1804-5. Then add the prepared Vagrant Box file to Vagrant, create a Vagrantfile and then start the virtual machine and then `ssh` into the box via Vagrant.

```
# clone the source code from the book to get the sample files
# git clone https://github.com/jhajak/Linux-text-book-part-1.git
cd Linux-text-book-part-1/files/Chapter-13/packer-build-templates
packer build ubuntu18045-vanilla.json

# Upon completion of the Packer build...
# Each build has a string representation of the day, month, year to make each
# filename unique, called epoch (your *.box name will be different)
vagrant box add ../build/ubuntu18045-vanilla-1574788560.box --name ubuntu-18045-vanilla
cd ../build
mkdir ubuntu-18045-vanilla
vagrant init ubuntu-18045-vanilla
# this command will show the *.box files that Vagrant knows about
vagrant box list
vagrant up
vagrant ssh
# exit the ssh session
vagrant halt
```

13.6 Secrets Management

One of the hardest parts of building software applications is managing **secrets**. Secrets can be anything from a username and a password, a token, or even keys from cloud services. The important part is that they are important. If you lose these secrets it could mean the end to a company. You also have to worry about invalidating them. If a person leaves, or rotates job, you don't want credentials to walk out the door with you. Also managing these secrets not just for security but for automation is also a critical part of the infrastructure.

In Linux distros as well as Packer, there are methods for dealing with secrets. The first obvious example how to we *seed* a root password for a MySQL server? If you install it, there is always a password prompt? This precludes you from automating the install.

All Debian based distros have a configuration database called DEBCONF. This can be used to preseed and answers you may have to installation questions that come via `apt-get`. Here is an example:

```
# This is sample code to add to a provisioner
# script that will pre-seed the password for a database.
export DEBIAN_FRONTEND=noninteractive
FIRSTPASS="mariadb-server mysql-server/root_password password ilovebunnies"
SECONDPASS="mariadb-server mysql-server/root_password_again password ilovebunnies"
echo $FIRSTPASS | sudo debconf-set-selections
echo $SECONDPASS | sudo debconf-set-selections
```

This example will set the answer to the root password prompt for MariaDB and or MySQL. In the above code the password is *ilovebunnies*. This is an automation improvement, but a security nightmare, as now our **root** password is hardcoded into our code and will then be placed in our GitHub repo for all to see. We can mitigate this by using ENV variables like this:

```
# run this on the command line and the value will be exported to all shells (or set this in your .bashrc)
export $DBPASS="ilovebunnies"
# run this in a shell script
export DEBIAN_FRONTEND=noninteractive
FIRSTPASS="mariadb-server mysql-server/root_password password $DBPASS"
SECONDPASS="mariadb-server mysql-server/root_password_again password $DBPASS"

echo $FIRSTPASS | sudo debconf-set-selections
echo $SECONDPASS | sudo debconf-set-selections
```

This is better but not the best as others on the system or any code that can read ENV variables can now read the password. Packer has a way to pass ENV variables from a config file. This is similar to how WordPress is configured. Adding these lines of code to your Provisioner in your Packer build template allows this:

```
{
  "type": "shell",
  "execute_command": "echo 'vagrant' | {{ .Vars }} sudo -E -S sh '{{ .Path }}'",
  "script": "../scripts/post_install_itmt430-github-db.sh",
  "environment_vars": [
    "DBPASS={{user `database-root-password`}}",
    "USERPASS={{user `database-user-password`}}",
    "ACCESSFROMIP={{user `database-access-from-ip`}}",
    "DATABASEIP={{user `database-ip`}}",
    "DATABASENAME={{user `database-name`}}",
    "DATABASEUSERNAME={{user `database-user-name`}}"
  ]
}
```

Packer has the ability to set ENV variables upon install. From the command line you pass an additional `--var-file=` command and Packer will load values from that file.

```
packer build --var-file=./variables.json ubuntu18045-vanilla-multi-drives.json
```

```
{
  "database-root-password": "foo",
  "database-user-password": "bar",
  "database-access-from-ip": "127.0.0.1",
  "database-ip": "127.0.0.1",
  "webserver-ip": "127.0.0.1",
  "database-name": "namegoeshere",
  "database-user-name": "database-username-goes-here"
}
```

There is a consideration here, if you add values to `variables.json` they will still be pushed to your Git repo and you

will have the same problem. When you need to do is create a template. Essentially the file, `variables-sample.json` is just that a template to show you what values you can enter. You can copy the file, change the name to `variables.json` for instance. Then in your Git repo, add an entry to the `.gitignore`. This file in the root of your Git repo will ignore all files you tell it to. This way you can distribute your template for secrets and passwords, but now you can retain a local copy that will be exposed via Git. Sample files are provided in the files > Appendix-D > packer-scripts folder.

13.6.1 How to Manage Secrets

But what happens when your secrets need to be managed by multiple people across a large enterprise, or even a large cloud enterprise? There is an opensource and enterprise grade product from HashiCorp called Vault. It does what is says, essentially cryptographically storing all the secrets you enter into a *vault*, then delegating access to these secrets via API (over HTTP) allowing for the implementation of policy and identity relating to accessing these secrets. The *vault* can then be attached or mounted into any system and each developer can access their secrets. Vaults use cases are as follows”

Vault tightly controls access to secrets and encryption keys by authenticating against trusted sources of identity such as Active Directory, LDAP, Kubernetes, CloudFoundry, and cloud platforms. Vault enables fine grained authorization of which users and applications are permitted access to secrets and keys. Vault can be integrated with other platforms as well, Active Directory, AWS IAM profile management, and other platforms.

13.6.1.1 Vault Integration With Packer

For our convenience, Packer has direct integration with Vault. Once Vault is installed an setup on your [local system](#) for instance by running the Vault agent you can simply read your secrets in the Packer Build Template, without the secret ever being seen by a person. There is more to say on this, but the reason we introduce it here is so that you can be exposed to safe practices from the beginning as well as deal with one of the major problems in IT, which is Secrets Management.

```
{
  "variables": {
    "database-root-password": "{{ vault `secrets/database-root-password` `database-root-password` }}",
    "database-user-password": "{{ vault `secrets/database-user-password` `database-user-password` }}",
    "database-access-from-ip": "{{ vault `secrets/database-access-from-ip` `database-access-from-ip` }}",
    "database-ip": "{{ vault `secrets/database-ip` `database-ip` }}",
    "webserver-ip": "{{ vault `secrets/webserver-ip` `webserver-ip` }}",
    "databaseslave-ip": "{{ vault `secrets/databaseslave-ip` `databaseslave-ip` }}",
    "cache-ip": "{{ vault `secrets/cache-ip` `cache-ip` }}",
    "salt": "{{ vault `secrets/salt` `salt` }}"
  }
}

{
  "type": "shell",
  "execute_command": "echo 'vagrant' | {{ .Vars }} sudo -E -S sh '{{ .Path }}'",
  "script": "../scripts/post_install_itmt430-github-db.sh",
  "environment_vars": [
    "DBPASS={{user `database-root-password`}}",
    "USERPASS={{user `database-user-password`}}",
    "ACCESSFROMIP={{user `database-access-from-ip`}}",
    "DATABASEIP={{user `database-ip`}}",
    "DATABASENAME={{user `database-name`}}",
    "DATABASEUSERNAME={{user `database-user-name`}}"
  ]
}
}
```

Packer provides the ability to execute inline Linux commands. This would be useful for instance in copying code from a Git repo into a Virtual Machine via making an SCP (Secure Copy) connection.

```
{
```

```

"type": "shell",
"execute_command": "echo 'vagrant' | {{ .Vars }} sudo -E -S sh '{{ .Path }}'",
"inline": [
  "mkdir -p /var/www/html",
  "cp -v ../github-repo/php /var/www/html",
  "cp -v ../github-repo/sql /home/vagrant/",
]
}

```

13.6.2 IT Orchestration

In looking at these tools, Vagrant, Packer, Preseed, and Kickstart, we begin to see a world of automation opening up to us. In a sense these technologies are the culmination of the Unix concepts of small tools doing one things—or shell scripts on steroids, so to speak. Each of these technologies is beyond the scope of this book, but here are some podcasts and links to learn more about them. Some relate to installing traditional virtual machines, some to bare metal installs, others assume cloud platforms.

- [SaltStack](#)
 - [Podcast](#)
- [Chef](#)
 - [Podcast](#)
- [Puppet](#)
 - [Podcast](#)
- [Ansible](#)
 - [Podcast](#)
- [CfEngine](#)
 - [Podcast](#)
 - [Dr. Mark Burgess](#) wrote a book about his research in IT called [In Search of Certainty](#)
- [Cobbler](#)
- [Terraform](#)
 - [HashiCorp Terraform Presentation](#)

13.7 Chapter Conclusions and Review

In this chapter we learned how a spread in technology led to a desire to automate and ease installation and configuration. A new generation of technology brought a new generation of tools. These tools are a part of what we called cloud-native, immutable infrastructure, and are the standard ways to deploy operating systems and hardware across the industry.

13.7.1 Review Questions

1. What is a common title given to IT workers who are responsible for the ongoing operations of applications and their environments?
 - a) saints
 - b) devs
 - c) devops
 - d) ops
2. What would describe Mitchell Hashimoto’s design goals in created Vagrant?
 - a) Automation
 - b) Separation of Duties
 - c) Profit
 - d) Inspection
3. What is the name of the tool originally built as an abstraction layer on top of VirtualBox to deploy virtual machines?
 - a) Packer
 - b) VirtualBox

- c) Terraform
 - d) Vagrant
4. What is the name of the tool originally built as a way to automate the installation of any operating system into an artifact?
 - a) Packer
 - b) VirtualBox
 - c) Terraform
 - d) Vagrant
 5. What year approximately was Vagrant introduced?
 - a) 2019
 - b) 2001
 - c) 2010
 - d) 2015
 6. Fill in the blank. Think of Vagrant as _____ between you and VirtualBox, Hyper-V, Docker, or even VMware desktop.
 7. What is the name of the file type Vagrant uses that contains an vmdk and and ovf?
 - a) *.vdi
 - b) *.vhd
 - c) *.box
 - d) *.zip
 8. Name the file that contains the configuration file for each Vagrant box file.
 9. What is the correct command to add the Vagrant Box centos/7?
 - a) `vagrant add box centos/7`
 - b) `vagrant box add centos/7`
 - c) `vagrant init centos/7`
 - d) `vagrant add centos/7`
 10. What is the command used to list all Vagrant Boxes being managed by Vagrant?
 - a) `vagrant list box`
 - b) `vagrant boxes list`
 - c) `vagrant box list`
 - d) `vagrant list`
 11. What is the correct command to initialize a Vagrant file for Vagrant Box named centos/7 that has already been added to the system?
 - a) `vagrant init`
 - b) `vagrant init centos/7`
 - c) `vagrant box add centos/7`
 - d) `vagrant init 7`
 12. What is the Vagrant command to start or turn on a Vagrant Box?
 13. What is the Vagrant command to restart a Vagrant Box?
 14. What is the Vagrant command to shutdown or poweroff a Vagrant Box?
 15. For Packer.io, what is the descriptive name of the json file used for building artifacts?
 - a) image template
 - b) machine.json
 - c) build template
 - d) provisioner
 16. What is the name of the stage that runs after the builder portion of a build template?

- a) imager
 - b) provisioner
 - c) vboxmanage
 - d) post-processor
17. What is the name of the stage that runs after building and provisioning of Packer artifacts is complete?
- a) imager
 - b) provisioner
 - c) vboxmanage
 - d) post-processor
18. If there is an error in any part of the Packer build command what will happen?
- a) nothing
 - b) an error will be logged but the process will continue
 - c) the command will terminate and any artifacts will be deleted
 - d) the user will be prompted
19. What is the generic name of the file that is provided to Packer to help it complete the manual question part of the install?
- a) secret file
 - b) answer file
 - c) question file
 - d) pxe file
20. What are the respective names of the Red Hat and Debian based answer files?
- a) jumpstart and preseed
 - b) kickstart and jumpstart
 - c) kickstart and preseed
 - d) Chef and Puppet

13.7.2 Podcast Questions

See the presentation at: <https://www.youtube.com/watch?v=xXWaEck9XqM>: The Container Revolution: Reflections After the First Decade by Bryan Cantrill.

1. ~0:30 Where does/did Bryan work, who recently bought that company, and what do they do?
2. ~1:33 What is the birth date of containers?
3. ~3:25 What was the next iteration of containers?
4. ~3:49 What is the purpose of a Jail?
5. ~5:10 What did Sun call their full application environment they created in 2002?
6. ~6:13 What is every application running on?
7. ~8:43 What did Amazon announce in 2006?
8. ~9:00 In 2006 what technology was Joyent using to run its Public Cloud? In 2006 what technology was Amazon using to run its Public Cloud?
9. ~9:25 What became de facto for the cloud?
10. ~11:18 What happens to the RAM when you give it to an operating system?
11. ~14:40 What does Joyent's Manta service allow you to do with containers and objects?
12. ~18:58 What command hadn't been created in 1986?
13. ~21:45 When did the world figure out containers and what was this product?
14. ~22:57 Why did the container revolution start with Docker?
15. ~24:07 Containers allow developers to do what?
16. ~26:00 What is Triton and what does it do?
17. ~31:42 What are the two approaches to the container ecosystem, and what is the difference?
18. ~33:25 What is the "Hashi" ethos?
19. ~37:00 What was the mistake that happened with the pilot-operated relief valve at 3 Mile Island?
20. ~39:05 According to the speaker, with container based systems, in what terms must we think in?
21. ~40:00 Why is scheduling containers inside of Virtual Machines a bad idea?
22. ~What are Joyent's thoughts regarding Virtual Machines in the application stack?

13.7.3 Lab

13.7.3.1 Part 1

Create a folder structure for 1 Ubuntu Bionic64 vagrant box and 1 CentOS 7 vagrant box. In each of these folders use the `vagrant init` command to create a `Vagrantfile`. Upon successfully executing the `vagrant up` command in both directories, take a screenshot of the output of the `vagrant box list` command.

13.7.3.2 Part 2

Run the packer json build templates for CentOS 7 and Ubuntu 18.04 from the textbook source code located in `files > Chapter 13 > packer-build-templates`, for each template execute `packer build centos-7-vanilla-json` and `packer build ubuntu18045-vanilla.json`. Once these Vagrant boxes are built, use the `vagrant box add` command to add them to your Vagrant system. Run the `vagrant init` command with the proper options to create a `Vagrantfile` and then run the `vagrant up` command to instantiate the box. Issue the command `vagrant ssh` and once logged in take a screenshot of the output of the command `free --giga` to list the amount of memory in the virtual machine.

Upon completion take a screenshot of the output of the `vagrant box list` command to show that these steps completed successfully.

13.7.3.3 Part 3

Edit the `Vagrantfile` for both Vagrant boxes to run at 3072 RAM (3 GB) each. Issue the command to reload and re-provision the virtual machines. Upon successfully issuing this command, issue the `vagrant ssh` command and again execute the `free --giga` command to show that the memory adjustment actually took place. Take a screenshot of the results.

13.7.3.4 Part 4

Utilize the `vagrant destroy` command. On each of the four Vagrant boxes that have been created in the previous steps, execute the command to install the Apache2 webserver:

- `sudo apt-get install apache2`
- `sudo yum install httpd`

Take a screenshot of the output of the `sudo systemctl status apache2` or `sudo systemctl status httpd` command. Exit the Vagrant box. Issue a `vagrant destroy` command, then a `vagrant up` command. Issue the `vagrant ssh` command and reissue the above `systemctl` commands to show that all 4 boxes have been destroyed and rebuilt. Take a screenshot of the results.

13.7.3.5 Footnotes

Chapter 14

Final Projects

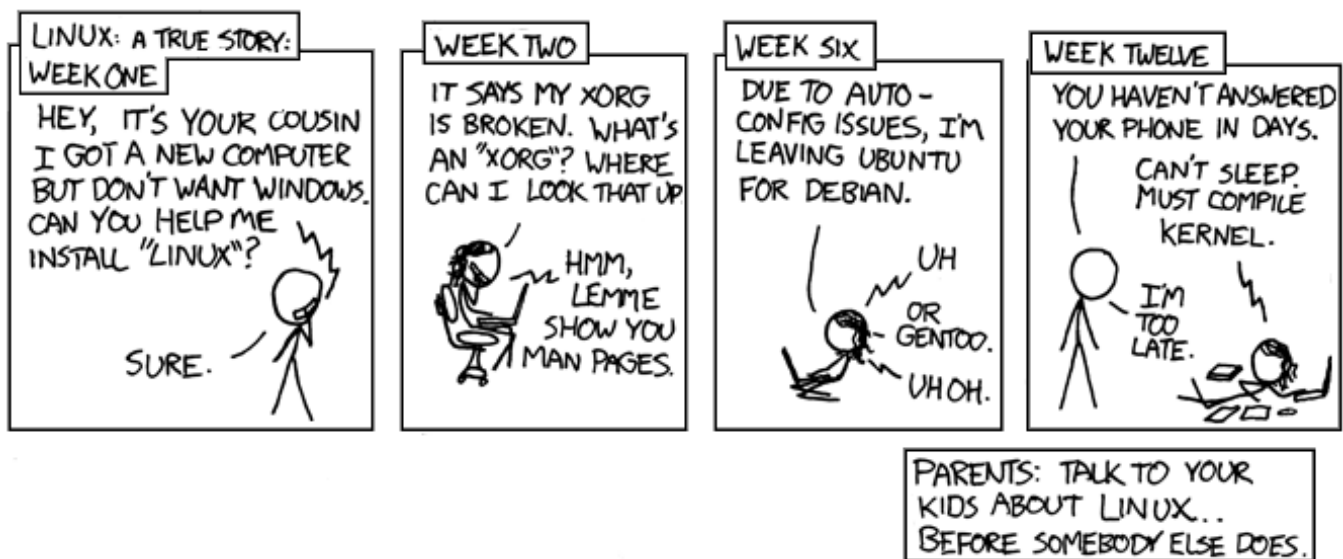


Figure 14.1: *Projects sometimes get out of hand...*

The list of final projects that applies all of the learned concepts and puts them together into a single deliverable.

14.1 Objectives

- Demonstrate the advantages of automation by building and deploying servers
- Demonstrate how to install software via a shell script
- Demonstrate how to preseed values for using MySQL database
- Display knowledge to create filesystems

14.2 Outcomes

At the conclusion of this project, you will have successfully demonstrated the basic installation and system administration concepts talked about in this book. Congratulations.

14.2.1 Part 1 - Using Ubuntu 18.04

This project will deploy and customize the existing Ubuntu 18.04 packer build template located in the files > Appendix-D directory. You will configure the provisioner file under the **scripts** directory to customize the Packer build script, `post_install_vagrant.sh`. The purpose is to deploy a single node [WordPress install](#). Upon completion of the build

task, you will import the Packer artifact from the build directory, the Vagrant box (virtual machine) that was created, that will have WordPress basic installation configured upon the first `vagrant up` command. The settings to make WordPress work automatically will be configured in the Packer **provisioner** shell script.

14.2.1.1 Packer Provisioner Requirements

The following software will be installed and configured in the provisioner script of Packer:

- 1) Using `deb-conf` pre-seed the root database password (yes you can hardcode the root password: `ilovebunnies`)
- 2) Install Apache2 webserver and MySQL server
- 3) From the WordPress.org tutorial, using `wget` retrieve the latest install zip file, install it, along with any needed pre-req software
- 4) Using `sed` and/or other tools, modify user variables in the WordPress configuration file to allow for WordPress to work without the need for a user to manually configure WordPress
- 5) Change the hostname to **wp-host-xyz**: xyz are your initials.
- 6) Change the timezone to UTC
- 7) During the Packer build, a second hard drive was attached. Install `btrfs-progs` and format that device.

14.2.1.2 Vagrant Requirements

The following actions need to take place to configure the Vagrant box:

- 1) Using Vagrant add the artifact generated from the output of the Packer command, name the box **wp**
- 2) Initialize the vagrant box using the name value as: `--name wp`
- 3) Modify the `Vagrantfile`, configure the system to use a *Private Network* with the IP: 192.168.33.100

14.2.1.3 Video Demonstration Requirements

The following actions need to be captured in a short video:

- 1) Use the command `vagrant up` command to launch the Vagrant box (this command may take a few minutes to run that is fine you can just let the video run)
- 2) Open a Web Browser and navigate to the IP address set in the `Vagrantfile`
- 3) Display the working WordPress “hello world” message
- 4) Log into WordPress and create a simple post entitled, “Final” with content that says “done.” Post it and return to the Welcome Screen showing the newly made post.
- 5) Close the browser tab
- 6) Issue the command `vagrant ssh` to ssh into the **wp** box
- 7) Display the changed hostname
- 8) Using the `timedatectl` command, display the timezone
- 9) Exit the ssh session
- 10) Issue a `vagrant halt` command – this is last command to capture – after end the video
- 11) Push all code used to construct this deliverable to GitHub (not the video) under the **final-project** folder

14.3 Deliverable

Create a folder in your private GitHub repo named **final-project** submit:

- All Packer build scripts, preseed/kickstarts, and provisioner shell scripts needed to run and build this application.
- Include all Vagrant files needed to start the Vagrant box
- To blackboard submit the URL to your GitHub repo as well as a URL to the video recording (preferably uploaded to your Google Drive account)

I recommend using the [OBS Studio project](#) for screen capture. It is a cross platform OpenSource solution used for major podcast production, but is simple enough that can be used for screen recordings. Deliver the Recording format using your school Google Drive account and enable access for me to retrieve it.

14.3.1 Points breakdown

Total point is 100.

- 80 points for the code
 - 10 points for the each of the 7 items required in Packer Provisioner Script Requirements
 - 10 points for completing all of the Vagrant Requirements
- 20 points for the video demonstrating the successful working code and the demonstration of a successful blog post

Chapter 15

Appendix A - Standards and Licenses

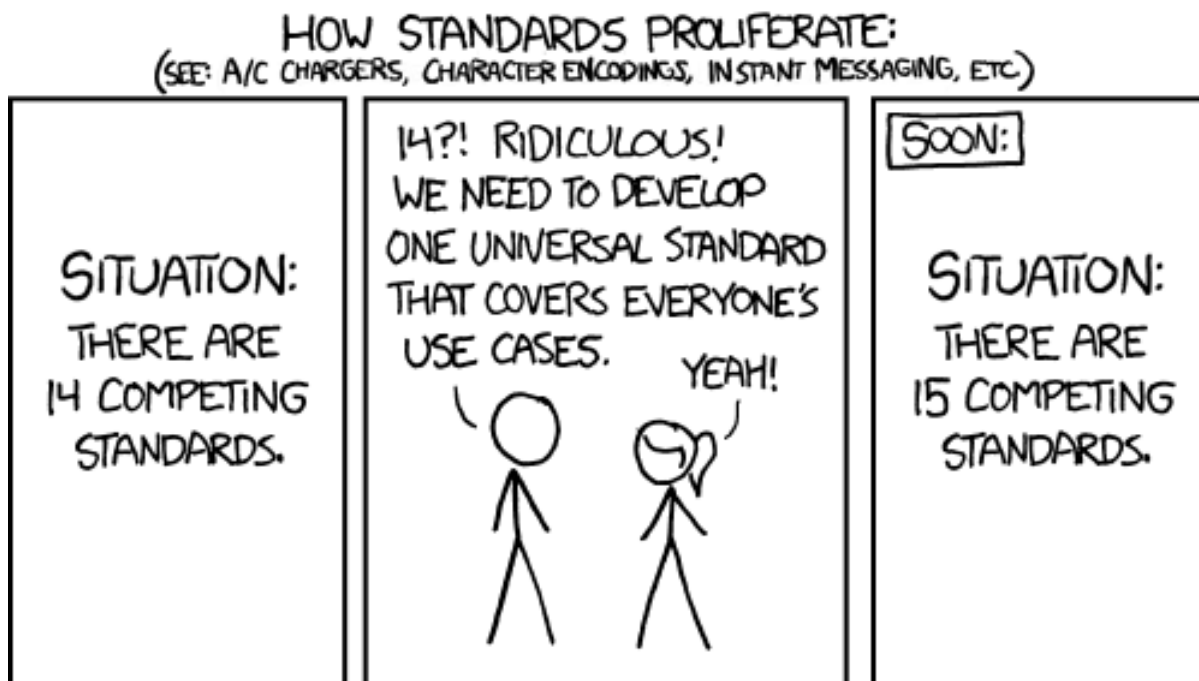


Figure 15.1: *How standards are made...*

15.1 POSIX Standard

The best place I found a short (trust me this is short) summary of what POSIX is and does was from an answer on StackOverflow, I encourage you to read the entire post and see if you can tell me what POSIX is in 3 sentences. From the discussion I highlighted this one answer that does the best.<http://stackoverflow.com/questions/1780599/i-never-really-understood-what-is-posix>.¹

15.1.1 Most important things POSIX 7 defines

1) C API

- Extends ANSI C with things like: networking, process and thread management, file IO, regular expressions, ...
– E.g.: write, open, read, ...

¹The answer was written by StackOverflow user [Siro-Santilli](#) The question was asked by StackOverflow user [Claws](#) and is licensed under CC BY-SA 3.0 (<http://creativecommons.org/licenses/by-sa/3.0/>). Question URL: <http://stackoverflow.com/questions/1780599/i-never-really-understood-what-is-posix>

- Those APIs also determine underlying system concepts on which they depend.
- Many Linux system calls exist to implement a specific POSIX C API function and make Linux compliant, e.g. `sys_write`, `sys_read`, ...
- Major Linux desktop implementation: glibc.

1) CLI utilities

- E.g.: `cd`, `ls`, `echo`, ...
- Many utilities are direct shell front ends for a corresponding C API function, e.g. `mkdir`.
- Major Linux desktop implementation: GNU Coreutils for the small ones, separate GNU projects for the big ones: `sed`, `grep`, `awk`, ... Some CLI utilities are implemented by Bash as built-ins.

1) Shell language

- E.g., `a=b; echo "$a"`
- Major Linux desktop implementation: GNU Bash.

1) Environment variables

- E.g.: `HOME`, `PATH`.

1) Program exit status

- ANSI C says 0 or `EXIT_SUCCESS` for success, `EXIT_FAILURE` for failure, and leaves the rest implementation defined.
- POSIX adds:
 - 126: command found but not executable.
 - 127: command not found.
 - 128: terminated by a signal.
- But POSIX does not seem to specify the 128 + [SIGNAL_ID rule used by Bash](#)

1) Regular expression

- There are two types: BRE (Basic) and ERE (Extended). Basic is deprecated and only kept to not break APIs.
- Those are implemented by C API functions, and used throughout CLI utilities, e.g. `grep` accepts BREs by default, and EREs with `-E`.
 - E.g.: `echo 'a.1' | grep -E 'a.[[:digit:]]'`

1) Directory structure

- E.g.: `/dev/null`, `/tmp`
- The [Linux FHS](http://refspecs.linuxfoundation.org/FHS_3.0/fhs/ch01.html “FHS” greatly extends POSIX.

1) Filenames

- `/` is the path separator
- NUL cannot be used
- `.` is `cwd`, `..` is `parent directory`
- minimum filename length that must be accepted is 14, 256 for full paths
- portable filenames should only contain: `a-zA-Z0-9._-`
- See also: [what is Posix compliance for filesystem?](#)

1) Command line utility convention

- Not mandatory, used by POSIX, but almost nowhere else, notably not in GNU. But true, it is too restrictive, e.g. single letter flags only.
- A few widely used conventions:
 - single dash means stdin where a file handle is expected: `-`
 - double dash terminates flags: `--`
- See also: [Are there standards for Linux command line switches and arguments?](#)

Who conforms to POSIX?

Many systems follow POSIX closely, but few are actually certified by the Open Group which maintains the standard. Notable certified ones include:

- AIX (IBM)
- HP-UX (HP)
- Solaris (Oracle)
- OSX (Apple)

Most Linux distros are very POSIX compliant, but not certified because they don't want to pay the compliance check fee.

15.1.1.1 Footnotes

Chapter 16

Appendix B - Answers for Review Questions

This section contains the questions and answers to the chapter review questions.

16.1 Chapter 01

NA

16.2 Chapter 02

NA

16.3 Chapter 03 - Review Questions

Hardware and Linux Installation

1. What is the term for the industry standard file format that is used to install a Linux distro?
 - a. ISO
2. What is currently the most common Linux install media type?
 - c. USB
3. What is the name of the most common tool used to create bootable Linux install media?
 - c. etcher.io
4. What is the technology that is inserted between ring 1 and ring 0 that enables virtualization? A: hypervisor
5. The operating system that the hypervisor resides on is called the _____ system? A: host
6. Hosted or desktop virtualization is called what type of hypervisor? A: Type II
7. Bare Metal or Native Virtualization is called what type of hypervisor? A: Type I
8. Each Linux installation distro provides a mechanism to compare what you downloaded with what you expected to download, what is that called?
 - b. check sum
9. What is the name of the driver package you can install in VirtualBox in order to enable features such as shared clipboard, larger screen resolution, and mouse pointer integration?
 - d. VirtualBox Guest Additions

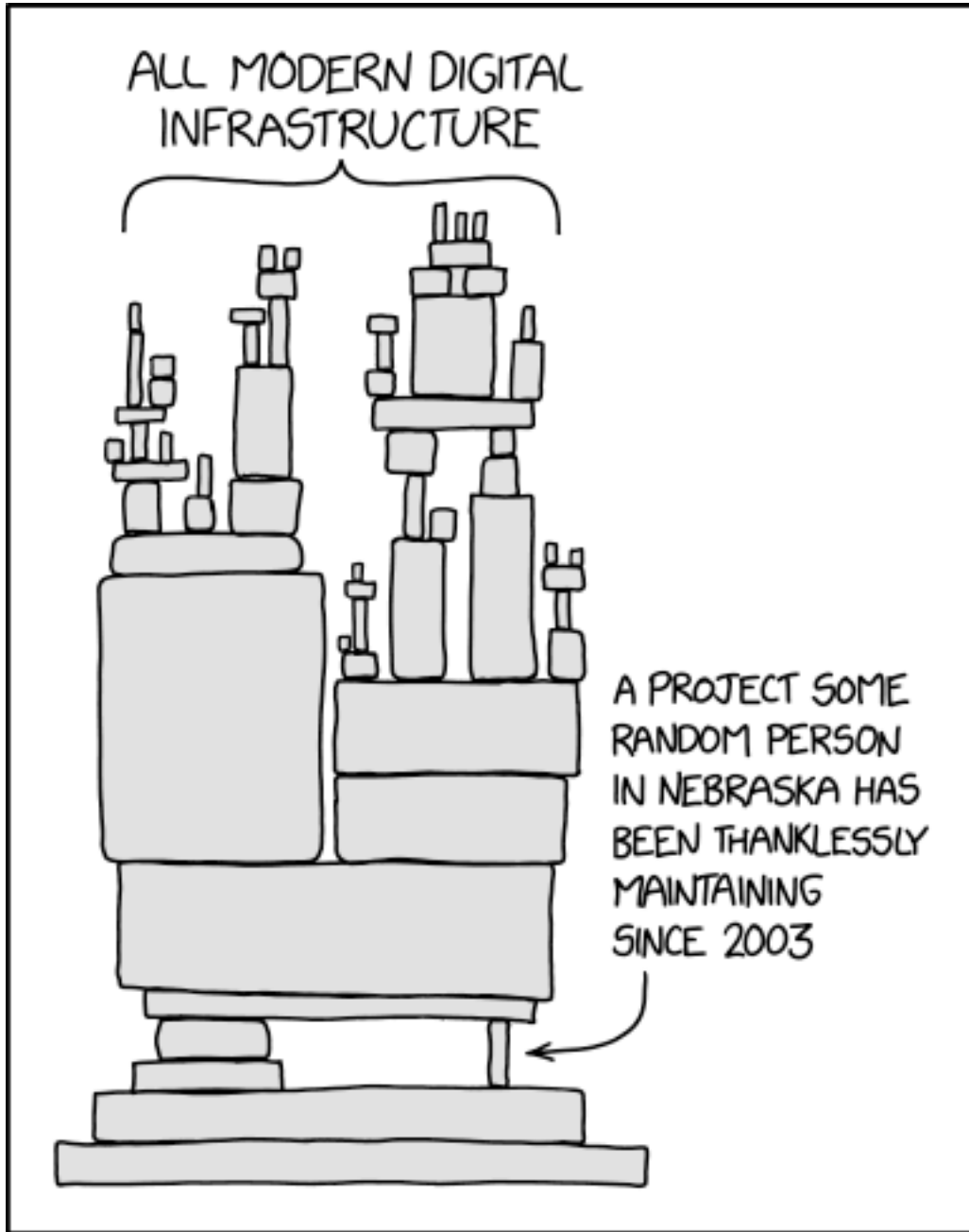


Figure 16.1: *One little thing holding everything up*

10. What is the name for a Linux distribution that runs in memory?
 - b. Live ISO
11. What feature doesn't dpkg handle/support?
 - b. Installing dependencies
12. What is the APT command to add an additional software repository in Ubuntu/Debian, named: `ppa:linux-libre/ppa`, to your APT system?
 - b. `sudo add-apt-repository ppa:linux-libre/ppa`
13. Which distro(s) supports the .deb package?
 - b. Debian Family
14. Which distro(s) supports the RPM package?
 - b. RedHat Family
15. We talked about using GCC to compile and install software, what was the other language/package manager discussed in the chapter?
 - c. Python
16. Describe the purpose of VirtualBox Guest Additions? The Guest Additions are designed to be installed inside a virtual machine after the guest operating system has been installed. They consist of device drivers and system applications that optimize the guest operating system for better performance and usability.
17. In APT, where are the additional source list fragments stored (file location)?
18. After building software from source and running the `./configure` command, what is the next step?
 - d. Run the `make` command
19. What is the name of the new package manager Ubuntu/Canonical has developed? `snap`
20. What is the name of the new package manager RedHat has developed? `flatpak`

16.4 Chapter 04 - Review Questions

Linux Desktop and GUI

1. What was the original and most popular Unix "Smart Terminal"?
 - b. VT-100
2. What is the three letter abbreviation still in use today in modern Linux to refer to "terminal devices"?
 - b. TTY
3. Why is the GNOME terminal and Windows `cmd.exe` terminal emulator screens 80 by 25 by default?
 - c. The developers of these technologies were seeking to emulate the popular VT-100 and VT-0220 terminals they used prior.
4. What is the key combo sequence you can hit to switch to a new virtual terminal in Linux?
 - b. `Alt + Ctrl + F1 - F7`
5. What is the name of the original Unix based GUI that came out of MIT in ~1984?
 - b. X
6. True or False – X was originally not opensourced by MIT in 1984
 - a. True
7. What is the model that the X server uses to render screens? Client Server
8. What is the name of the successor GUI compositor to X being created by the X.org foundation?

- c. Wayland Project
- 9. What is the name of Ubuntu's own GUI compositor replacement for X? (Just the name.)
 - a. Mir
- 10. What are the 3 types of Linux window managers? Compositing, Stacking, and Tiling
- 11. There are 4 major Linux desktop environments in use today: which grouping is correct?
 - d. KDE, GNOME, Xfce, LXDE
- 12. What is the name of the windowing toolkit that KDE uses? (Two letters)
 - a. Qt
- 13. What is the name of the windowing toolkit that GNOME uses? (just 3 letters, no plus sign.)
 - a. GTK+
- 14. What is the name of the founder of the GNOME project?
 - a. Miguel de Icaza
- 15. When the GNOME 3 desktop environment was released in early 2012, many people were unhappy that many changes were made. There were 3 major projects started to either preserve GNOME 2 or to modify GNOME 3 significantly – what are the names of those projects?
 - c. Unity, Mate, Cinnamon
- 16. When released in 2006, what was the main development goal of the LXDE desktop environment?
 - c. Energy saving and extremely fast
- 17. Which of these statements are true in regards to Linux desktop environments? (choose all that apply).
 - a. Desktop environments have a File Manager
 - b. Desktop Environments have start or action buttons and task and notification bars
 - c. Desktop environments have a changeable but consistent look-and-feel
 - d. Desktop environments have system configuration tools and user applications built-in
- 18. What is the default desktop environment for the latest Fedora desktop? (Name and version)?
 - a. GNOME 40
- 19. What is the name of the button on the upper left hand corner of the GNOME 3 desktop that you use to “launch applications?”
 - a. Activities
- 20. What is the name of the default Ubuntu desktop environment? GNOME

16.5 Chapter 05 - Review Questions

The Linux Filesystem, Path, Shell, and File Permissions

1. What is the numeric value of a file with the permissions `rwxr-r-`?
 - d. 744
2. What is the numeric value of a file with the permissions `rw---`?
 - c. 600
3. What is the numeric value of a file with the permission `rwX-r-X-X`?
 - b. 755
4. What is the name of the command you use to list the contents of a directory?
 - d. `ls`

5. What is the name of the command you use to display the content of a file to the screen?
 - d. cat
6. What is the name of the command you use to display the content of a file to the screen that allows you to page up and down?
 - b. less
7. Every shell command has three components:
 - b. Command, options, arguments
8. What does the user use to issue commands to the kernel?
 - d. Shell
9. What is the name of the GNU shell that is standard across all Linux Distros?
 - d. bash
10. Based on the ls command, what is the option to do a long listing?
 - d. ls -l
11. What is the command you can use to find out additional usage information about a shell command?
 - b. man
12. Which of these directories is not part of the LSB LFH?
 - c. temp
13. The Linux Filesystem is an upside down what?
 - c. tree
14. What is the character name of the top of the Linux Filesystem tree?
 - a. /
15. Everything (directories, files, devices. in Unix/Linux is a what?
 - a. file
16. What is the name of the Unix system standard developed in ~1985 that defines at a minimum what a certified Unix based system must support?
 - c. POSIX
17. What is the directory where all configuration files are stored in Linux?
 - c. etc
18. What is the directory where all system binaries are stored?
 - a. bin
19. What is the absolute path of a user's home directory, assuming the user is named controller?
 - a. /home/controller
20. True or False - Lennart Poettering supports POSIX
 - a. False

16.6 Chapter 06 - Review Questions

Shell Meta-Characters, Pipes, Search, and Tools

1. What is the name for characters that have special meanings in the Linux shell?
 - d. shell meta-characters

2. Assume your pwd is ~. If you wanted to list every directory only that started with the letters “Do” what would be the command?
 - d. `ls -l Do*`
3. In figure 6.2 in Chapter 06 which of the 3 blue boxes is the step where shell meta-characters are transformed into text?
 - d. Expansion
4. Which meta-character allows you to string commands together regardless of the successful execution of the previous command?
 - b. ;
5. Which meta-character allows you to string commands together but will exit if one of the commands fails?
 - a. `&&`
6. Which meta-character is the wildcard (0 or more matches)?
 - d. *
7. Which meta-character is the single character wildcard?
 - a. ?
8. Square braces [] indicate sets or _____ of characters to be processed
 - d. ranges
9. If you wanted to use brace expansion and create a series of nine files named: file1, file2, file3, etc etc all at once—what command would you use? (type the full command using touch .
 - d. `touch file{1..9}`
10. If you wanted to assign the value of /etc/alternatives/java to a shell variable named JAVA_HOME—what would be the proper syntax?
 - c. `JAVA_HOME=/etc/alternatives/java`
11. What would be the proper syntax to display the content of a variable named JAVA_HOME in the shell?
 - d. `echo $JAVA_HOME`
12. There are 3 standard I/O devices in a Linux system, standard in, standard out, and _____
 - d. standard error
13. Standard In is what device by default?
 - d. keyboard
14. Standard Out is what device be default?
 - b. screen
15. What meta-character can you use to redirect standard out to a file? (Choose all that apply.
 - a. >
 - b. »
16. What meta-character is used to redirect the standard output of one command as the standard input of another command?
 - d. |
17. Which command is a shortcut to display the kernel’s output messages?
 - d. `dmesg`
18. Which command is used to search within files using textual filter patterns?

- d. grep
- 19. Which command can be used to count lines that are in a text file?
 - b. wc
- 20. Which command can be used to find unique line occurrences in a text file?
 - b. uniq

16.7 Chapter 07 - Review Questions

Introduction to Linux Editors, Introduction to Shell Scripts, and User Profiles

1. What are the two main representatives of stream editors
 - d. vi and Emacs
2. Which family of editors came first?
 - d. Stream Editors
3. Emacs at its core is what?
 - c. An interpreter for Emacs Lisp
4. Who created the vi editor and in what year?
 - c. Bill Joy, 1979
5. Which of the following sequences of the history of vi is correct?
 - b. ed -> em -> ex -> vi -> vim
6. What are the three modes in vi? (separate each answer via a comma)
 - a. command, insert, ex
7. What is the key you use in vi to transition between COMMAND MODE and INSERT mode? (Just three letters)
 - a. ESC
8. What command sequence (key) in vi will add text to the right of the current cursor position? (just the letter)
 - a. a
9. What command sequence (key) in vi will move you to the beginning of the next word? (just the letter)
 - a. w
10. What command sequence in vi will delete a single line based on the current cursor position? (just the letters)
 - a. dd
11. What command sequence in vi will delete 10 lines from the current cursor position? (just the numbers and letters)
 - a. 10dd
12. Which command in ex mode (vi) will save the current file you are working on and exit the vi editor? (include the :)
 - a. :wq
13. In the log file u_ex150911_.log what would be the ex command to search forward for occurrences of xmlrpc.php? (include the forward slash)
 - a. /xmlrpc\.php

14. Assuming your pwd is Linux-text-book-part-I and you have loaded Chapter-02.chapter-02.md into vi, what would be the ex mode command to replace all occurrences of linux with Linux?
 - a. 1,\$s/linux/Linux/g
15. Assuming your pwd is Linux-text-book-part-I and you have loaded Chapter-02.chapter-02.md into vi, what would be the ex mode command to replace all occurrences of Linux with GNU/Linux? (remember to escape the /)
 - a. 1,\$s/Linux/GNU\Linux/g
16. Assuming the your pwd is Linux-text-book-part-I and you have loaded Chapter-02.chapter-02.md into vi, what would be the ex mode command to remove all occurrences of the word Windows?
 - a. 1,\$s/Windows//g
17. Assuming a file name topsecret.sh has a permission of 644 - what is the shortcut to give the owner of the file permission to execute the script?
 - a. chmod u+x topsecret.sh
18. Assuming a file named moretopsecret.sh has a permission of 757 - what is the shortcut to remove all permissions from the the other group?
 - a. chmod o-rwx moretopsecret.sh
19. What is the correct sequence of profile inheritance?
 - a. /etc/profile -> ~/.bash_profile or ~/.bash_login or ~/.profile -> ~/.bashrc
20. What is the command to display the contents of the PATH system variable on the command line?
 - b. echo \$PATH

16.8 Chapter 08

- 1) True or False The Bash shell scripting language has traditional language constructs like C or Java? True
- 2) What meta-character do you use to access the content of a shell variable?
 - a. \$
- 3) When assigning the standard output of a command to a variable what characters do you encase the command in?
 - a. “
- 4) True or False - You can include shell meta-characters inside of two backticks `` - example:ANS='ls -l test[1-5]'" True
- 5) Which command will list the names of any file that matches these names: file1.txt file2.txt file3.txt file4.txt and send the content of that output to a variable named DIR?
 - d. "DIR=\ls -l ./test[1-4].txt"
- 6) Which of these are valid commands in the first line of a shell script? (Choose any - assume any paths are valid paths to executables)
 - a. #!/bin/bash
 - b. #!/usr/local/bin/bash
 - c. #!/bin/ksh
- 7) If you stored the output of the command hostname into a variable named sys-hostname, what would be the command to print the content to the screen?
 - d. echo \$sys-hostname
- 8) What is the name of the command to print out all the predefined system variables? printenv

- 9) What is the name of the command that allows you to take stdout of a command and insert the lines of output into an array?
- d. `mapfile`
- 10) Which of these is a valid command to take the output of this `find` command and assign the contents to an array? (Assume the array name has already been declared. Choose one)
- c. `mapfile -t SEARCHARRAY <<(find ~ -name mozilla*)`
- 11) Which below is a valid command to find the `LENGTH` of an array?
- a. `${#SEARCHARRAY[@]}`
- 12) Based on this shell script and positional parameters, what would the command be to print out the first positional parameter after the script name? `./delete-directory.sh ~/Documents/text-book Jeremy`
- b. `echo $1`
- 13) Based on this shell script and positional parameters, what would the command be to print out the entire content of the positional parameter array? `./delete-directory.sh ~/Documents/text-book Jeremy`
- d. `echo $@`
- 14) Based on this shell script and positional parameters, what would the command be to print out the `LENGTH` of the positional parameter array? `./delete-directory.sh ~/Documents/text-book Jeremy`
- a. `echo $#`
- 15) In a Bash IF statement, what is the name for the pre-made test conditions?
- a. Primaries
- 16) All values in a Bash IF statement are of what data type by default?
- b. `STRING`
- 17) Which of these answers will execute a shell script named `~/backup.sh` at 2 am every night of the week?
- c. `* 2 * * * ~/backup.sh`
- 18) Which of these answers will execute a shell script named `~/clean-directory.sh` every 15 minutes?
- b. `*/15 * * * * ~/clean-directory.sh`
- 19) Which of the crontab builtins would you use to execute a cron job 1 time a year on midnight of January 1st? The name of the script is `~/give-free-cash-to-students.sh`
- d. `@yearly ~/give-free-cash-to-students.sh`
- 20) What is the name of the control structure that allows you to incrementally through the contents of an array?
- d. `FOR`

16.9 Chapter 09

Review Questions Chapter 09

- 1) What user account has superuser privilege in Linux?
- d. `root`
- 2) Which command do you use to temporarily elevate your user's privilege to the superuser (`root`)?
- b. `sudo`
- 3) How can I display the content of a file named `topsecret.txt` that has permissions `000`?
- c. `sudo cat topsecret.txt`
- 4) What license is the `sudo` application under?

- d. ISC
- 5) Which operating system doesn't have an active root account by default?
 - b. Ubuntu
- 6) What is the name of the file where sudo privilege are kept?
 - d. `/etc/sudoers`
- 7) What is the name of the command used to modify `/etc/sudoers` to grant a new user sudo privilege?
 - c. `visudo`
- 8) Based on this line in `/etc/sudoers` - `%meninblack ALL=(ALL:ALL) ALL` - what does the first value by the % mean?
 - a. Name of a group
- 9) In the `/etc/sudoers` file - what does this line mean: `RMS ALL=(root) NOPASSWD: ALL`
 - d. The user RMS has sudo permissions and access to all commands, and requires no password to elevate to the sudo user
- 10) When using the `su` command to switch from a regular user account to the root user account, what do you type to return to the standard user account?
 - b. `exit`
- 11) What command would you use to edit the file at this location: `/var/www/html/index.html`?
 - b. `sudo vi /var/www/html/index.html`
- 12) On a Linux system, which directory are all the traditional system (non-systemd) logs kept in?
 - d. `/var/log`
- 13) Under `systemd` and `journald` where are the logs kept?
 - d. Trick question - as logs are stored in a binary format and retrieved via `journalctl`
- 14) What is the command you use to query the system logs in `systemd`?
 - c. `journalctl`
- 15) How would you filter the `systemd` log based on time?
 - a. `journalctl --since=yesterday`
- 16) What file would you edit to change the `systemd` `journald.conf`?
 - d. `/etc/systemd/journald.conf`
- 17) What command provides a dynamic real-time view of a running system?
 - a. `top`
- 18) Debian based distros have an additional command to abstract the process to add users to the system - what is it?
 - c. `adduser`
- 19) What command would be used to modify a user account settings and add them to the sudo users group (user is named `controller`)?
 - b. `sudo usermod -aG sudo controller`
- 20) Which below are valid `useradd` commands? (Choose all that apply)
 - a. `sudo useradd -c "User for spring class" -d "/home/export/controller" -G sudo -s /bin/ksh -m controller`
 - b. `sudo useradd controller`
 - c. `sudo useradd -G sudo -s /bin/ksh -m controller`

d. `sudo useradd -c "User for spring class" -G sudo -m controller`

16.10 Chapter 10

Chapter 10 Review Questions

- 1) What is the name of *beep* sound heard in the initial boot of a PC (assume you are using BIOS not UEFI)?
 - b) POST
- 2) What is the name of the GNU software that is the first software program that runs when a computer with Linux installed starts?
 - c) GRUB
- 3) In what Linux directory is the kernel and initrd image stored?
 - c) /boot
- 4) What is the name of the pre-kernel gzip file located in /boot that helps the kernel load?
 - b) initrd
- 5) Where is the file location where the GNU Grub configuration is stored that a user would edit?
 - b) /etc/default/grub
- 6) In the /etc/default/grub file, which of these options below would I edit to display the *splash* screen on boot so kernel messages are displayed?
 - a) GRUB_CMDLINE_LINUX_DEDFALT
- 7) What is the command to make changes to /etc/default/grub permanent?
 - c) `sudo update-grub`
- 8) Under SysVinit - what is the ancestor process that launches first and every other process is started by it?
 - c) `init`
- 9) Under SysVinit - what runlevel is considered multi-user command-line only?
 - c) 3
- 10) Under SysVinit - what runlevel is considered multi-user GUI only?
 - d) 5
- 11) Which company created the Upstart init system as an improvement of SysVinit?
 - d) Ubuntu
- 12) What is the name of the init system that has replaced SysVinit in every single major Linux distribution (Not including Devuan and Gentoo Linux)?
 - b) `systemd`
- 13) What is the name of the command you use in `systemd` to inspect, start, stop, and modify process states?
 - d) `systemctl`
- 14) What would be the command to disable (make the service not start at boot time) the `httpd` service on Fedora using `systemd`?
 - d) `sudo systemctl disable httpd.service`
- 15) What is the Linux command to inspect processes (not part of `systemd`)?
 - b) `ps`
- 16) SysVinit used the concept of PIDs and PPIDs—what did `systemd` replace these with?

- c) cgroups
- 17) What is the signal name for a kill -2 command?
 - b) SIGINT
- 18) The /proc filesystem provides you what? (choose all that apply)
 - a) Provides you a file based interface to the processes that are running on your system
 - b) It can be regarded as a control and information centre for the kernel
- 19) What command can be used to list all the pci devices attached to your system?
 - d) lspci
- 20) What is the runlevel target that has a single user only as root, using no password: commonly called single-user mode?
 - d) runlevel1.target

16.11 Chapter 11

Chapter 11 Review Questions

- 1) What is the `fdisk` program?
 - a) a dialog-driven program for the creation and manipulation of partition tables.
- 2) What is the default VirtualBox disk type?
 - a) VDI
- 3) After attaching a new virtual disk, what is the next step?
 - a) partitioning
- 4) Which command will print out currently all the block devices, their device name, and their partitions in a nice tree based format.
 - b) lsblk
- 5) `fdisk` is part of what package?
 - c) utils-linux
- 6) What would be the name of the second SATA disk attached to your system?
 - b) sdb
- 7) What is the name of the first native Linux filesystem released in 1992?
 - d) ext
- 8) What is the name of the current default Linux Filesystem?
 - d) ext4
- 9) Ext4 breaks up data into _____, which is the smallest sized piece of data that can be read or written?
 - c) blocks
- 10) If you use the ext2 filesystem and choose a 4 KiB block, what is the maximum filesystem size?
 - b) 16 TiB
- 11) What is the name of the maintainer of the ext4 filesystem?
 - b) Theodore Ts'o
- 12) What is the name of the filesystem that the ext4 maintainer, Theodore Ts'o, is recommending to replace ext4?
 - b) Btrfs

- 13) What is the name of the filesystem that RedHat adopted on their RHEL 7 platform to replace ext4 and support better performance on large filesystems?
 - b) XFS
- 14) Which is the correct command needed to install on Ubuntu to be able to create XFS filesystems?
 - a) `sudo apt-get install xfsprogs`
- 15) What is the name of the combined filesystem and logical volume manager designed by Sun Microsystems?
 - c) ZFS
- 16) Which is the correct command for making an ext4 filesystem on a partition `/dev/sdb1`?
 - b) `sudo mkfs.ext4 /dev/sdb1`
- 17) Which is the correct command to mount an ext4 filesystem, `/dev/sdb1` on a mount point `/mnt/data-drive-2`?
 - c) `sudo mount -t ext4 /dev/sdb1 /mnt/data-drive-2`
- 18) Which file contains the mountpoints that will be mounted automatically at boot?
 - c) `/etc/fstab`
- 19) What is the command used to create a LVM physical volume?
 - a) `pvcreate`
- 20) What is the command used to create a LVM volume group?
 - a) `vgcreate`

16.12 Chapter 12

1. Using the ip2 suite of tools, which command would show your IP address?
 - c. `ip address show`
 - d. `ip a sh`
1. Using the ip2 suite of tools, which command would show your routing table?
 - b. `ip route`
1. What tool could you use to establish if a server is responding to requests?
 - b. `ping`
1. What is the purpose of a NETMASK?
 2. What is the purpose of DNS?
 3. What is a valid class C address?
 - c. 192.168.1.0
1. If you had a network with a CIDR block of `/23` how many IP addresses would you have control of?
 - c. 512
1. If you had a CIDR block of `/24` and a network address of 192.168.1.0, how many IP addresses would you have?
 - d. 256
1. How does CIDR block addressing differ from Class based networking (A-D)?
 2. What tool is used to release a dhcp address from the command line?
 - c. `dhclient -r`
1. using the ip2 suite, What tool can be used to monitor and examine all current local ports and TCP/IP connections?
 - a. `ss`

1. Where are your network card settings located on Ubuntu while using network manager? `/etc/network/interfaces`
2. Where are your network card settings located on CentOS/Fedora using network manager? `/etc/sysconfig/network-scripts`
3. Where are your network card settings located on Ubuntu 18.04 using netplan? `/etc/netplan/config.yaml`
4. What are the two major opensource webservers?
 - a. Apache, Nginx
1. What are two related and major opensource relational databases?
 - b. MariaDB and MySQL
1. Name one major No-SQL database mentioned in this chapter? MongoDB
2. What is the file location that the system uses as a *local DNS* for resolving IPs?
 - b. `/etc/hosts`
1. What is the name of the file that you would place in your home directory, that allows you not to have to type your login password for a MySQL database?
 - d. `~/.my.cnf`
1. Before systemd, NIC interface naming schemes depended on a driver based enumeration process: they switched to a predictable network interface names process that depends on what for the interface names?
 - b. interface names depend on physical location of hardware (bus enumeration)

16.13 Chapter 13

1. What is a common title given to IT workers who are responsible for the ongoing operations of applications and their environments?
 - d) ops
2. What would describe Mitchell Hashimoto's design goals in created Vagrant?
 - a) Automation
3. What is the name of the tool originally built as an abstraction layer on top of VirtualBox to deploy virtual machines?
 - d) Vagrant
4. What is the name of the tool originally built as a way to automate the installation of any operating system into an artifact?
 - a) Packer
5. What year approximately was Vagrant introduced?
 - c) 2010
6. Fill in the blank. Think of Vagrant as _____ between you and VirtualBox, Hyper-V, Docker, or even VMware desktop an abstraction layer
7. What is the name of the file type Vagrant uses that contains an `vmdk` and `ovf`?
 - c) `*.box`
8. Name the file that contains the configuration file for each Vagrant box file.
9. What is the correct command to add the Vagrant Box `centos/7`?
 - b) `vagrant box add centos/7`
10. What is the command used to list all Vagrant Boxes being managed by Vagrant?
 - c) `vagrant box list`

11. What is the correct command to initialize a Vagrant file for Vagrant Box named `centos/7` that has already been added to the system?
 - b) `vagrant init centos/7`
12. What is the Vagrant command to start or turn on a Vagrant Box? `vagrant up`
13. What is the Vagrant command to restart a Vagrant Box? `vagrant reload`
14. What is the Vagrant command to shutdown or poweroff a Vagrant Box? `vagrant halt`
15. For Packer.io, what is the descriptive name of the json file used for building artifacts?
 - c) build template
16. What is the name of the stage that runs after the builder portion of a build template?
 - b) provisioner
17. What is the name of the stage that runs after building and provisioning of Packer artifacts is complete?
 - d) post-processor
18. If there is an error in any part of the Packer build command what will happen?
 - c) the command will terminate and any artifacts will be deleted
19. What is the generic name of the file that is provided to Packer to help it complete the manual question part of the install?
 - b) answer file
20. What are the respective names of the RedHat and Debian based answer files?
 - c) kickstart and preseed

16.14 Chapter 14

TBA

Chapter 17

Appendix C - Markdown Code for Podcasts, Labs, and Review Questions

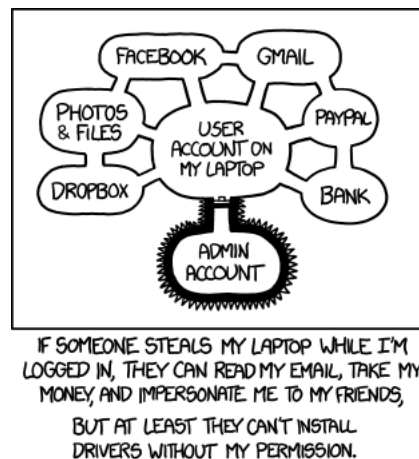


Figure 17.1: *Security is a strange term...*

This appendix includes all the review questions, labs, and podcast questions for each chapter in a single convenient place.

17.1 Chapter 02

17.1.1 Review Questions

Either individually, as a class, or get into groups and watch the documentary movie [Revolution OS](https://www.youtube.com/watch?v=jw8K460vx1c) - <https://www.youtube.com/watch?v=jw8K460vx1c> made in 2001. The film includes interviews with many of the names that were discussed in this chapter. Watch this movie and answer the questions below or via the online assignment provided for you by the instructor.

1. Based on the movie's tone and rhetoric - why do you think there was an anti-Microsoft tone at the time of the movies making (~2001)? (You may need to research [Microsoft anti-trust case](#)).
2. When Bill Gates wrote his 1976 "*Open Letter to Hobbyists*", was he justified in his complaint? Why or why not?
3. Would Richard Stallman enter into a discussion on which is a **better** product: Microsoft Word or LibreOffice Writer? Why or why not? Would Eric S. Raymond enter into a discussion on which is a **better** product: Microsoft Word or LibreOffice Writer? Why or why not?
4. Why did Bruce Perens help write the Open Source Definition / Debian Social Contract Standard?

5. What were the two commercial Linux companies featured in the movie (Note-one does not exist any longer)?
6. What is Red Hat Linux's stock price today compared to the the price listed in the movie? What is VA Linux's stock price today compared to the movie? (Hint VA Linux was sold and now belongs to another company, find that company's stock price.)
7. According to Eric S. Raymond what was the major application that needed to "flip" for opensource to become a viable enterprise solution?
8. What was the first major commercial company to opensource a key product? What did that product eventually become?
9. How does Richard Stallman react at the end of the movie to the success of the Linux kernel to the exclusion of the GNU toolchain?
10. What is the main difference between "*Free Software*" and "*Open Source Software*"?

17.1.2 Podcast Questions

Listen to the Podcast at <https://twit.tv/shows/floss-weekly/episodes/500>

- 7:21 Who is Vicky Brasseur and what does she do?
- 8:45 What has changed the most in regards to the term OpenSource in the last 25+ years?
- 10:00 What is assumed to now be the default development pattern for new software?
- 13:00 According to Randall, when a company publishes its code as OpenSource, what do they gain?
- 15:57 What do companies struggle with when they decide to opensource their codebase and core-products and how do they solve it?
- 19:00 What are some of the OpenSource companies?
- 19:35 What are the majority of OpenSource companies strategies to make money?
- 22:40 What do companies need to figure out about Free and OpenSource?
- 24:25 What is the term "Yak-Shaving" mean?
- 27:25 Who is Vicky's new book targeting?
- 28:03 What is the book about?
- 33:25 What is the mistaken impression about contributing to OpenSource Software?
- 37:40 What do you need to read before contributing to an OpenSource project?
- 43:00 What does Vicky believe is important for a project to have and to enforce for a community?
- 48:01 What is the myth about users of proprietary operating systems (Windows and MacOS)?
- 49:00 Vicky used Linux for 10 years, what drove her away from it?

17.1.3 Lab

17.1.3.1 Activity 1

Most of the popular and functional pieces of software you use everyday involve Free and OpenSource. Choose 2 case studies from <https://highscalability.com> and write a review of the company's architecture based the listed items.

You can find them at <https://highscalability.com/blog/category/example> or look on the High Scalability website on the lower right hand side for the "All Time Favorites" header to find some of the more popular services. Its best to find a company that you use or support.

Answer these questions (not all of the Answers are in each case study!)

1. What market does that company serve? (What do they do?) And have they always served that market?
2. What Operating System(s) are used?
3. What programming languages/frameworks are used?
4. What storage and what database technologies are used?
5. What is the current stock price and what was the IPO of the company? (if traded publicly.)
6. What major obstacle (cost, system performance, QPS, etc, etc) was the company trying to overcome by implementing this technology stack?
7. What can you learn from this article relating to technology and infrastructure?

17.1.3.2 Activity 2

Read these four articles. It's a commentary on the Opensource license changes, a response from one of the CEOs, and then a reply to the response.

- 1) <http://dtrace.org/blogs/bmc/2018/12/14/open-source-confronts-its-midlife-crisis/>
- 2) <https://medium.com/@jaykreps/a-quick-comment-on-bryan-cantrills-blog-on-licensing-8dccee41d9e6/>
- 3) <http://dtrace.org/blogs/bmc/2018/12/16/a-eula-in-foss-clothing/>
- 4) <https://medium.com/@adamhjk/goodbye-open-core-good-riddance-to-bad-rubbish-ae3355316494>
 - i) <https://sfosc.org/docs/business-models/free-software-product/>

Answer these questions with a few short sentences:

- What is Bryan Cantrill's initial main point in the first article?
- Do you agree with him, why or why not?
- What is Jay Kreps response/contention in the second article?
- Do you agree with him, why or why not?
- What is the main point of Bryan Cantrill's rejoinder in the third article?
- Do you agree with him, why or why not?
- What is Adam Jacob's opinion on running a company with an opensource product?
- What is the solution in your opinion?

17.1.3.3 Activity 3

- Sign up for a GitHub ID at [GitHub.com](https://github.com)
 - If you have one already no need to sign up for another one
- See Appendix-E assignment for installing Git on your computer

17.2 Chapter 03

17.2.1 Review Questions

- 1) What is the term for the industry standard file format that is used to install a Linux distro?
 - a. ISO
 - b. ZIP
 - c. Disk-ISO
 - d. Distro
- 2) What is currently the most common Linux install media type?
 - a. CD-ROMs
 - b. Network based installs
 - c. USB
 - d. Thunderbolt
- 3) What is the name of recommended tool used to create bootable Linux install media?
 - a. Pendrive Linux
 - b. etcher.io
 - c. UNetbootin
 - d. Image Magick
- 4) What is the technology that is inserted between ring 1 and ring 0 that enables virtualization?
- 5) The operating system that the hypervisor resides on is called the _____ system?
- 6) Hosted or desktop virtualization is called what type of hypervisor?
- 7) Bare Metal or Native Virtualization is called what type of hypervisor?
- 8) Each Linux installation distro provides a mechanism to compare what you downloaded with what you expected to download, what is that called?

- a. mount point
 - b. checksum
 - c. receipt
 - d. mdsum
- 9) What is the name of the driver package you can install in VirtualBox in order to enable features such as shared clipboard, larger screen resolution, and mouse pointer integration?
- a. Kernel modules
 - b. Kernel drivers
 - c. VirtualBox extensions
 - d. ISO extensions
- 10) What is the name for a Linux distribution that runs in memory?
- a. Rapid CD
 - b. Live ISO
 - c. Install Disk
 - d. Trick question
- 11) What feature doesn't dpkg handle/support?
- a. Dependencies
 - b. Installing Dependencies
 - c. Versioning
 - d. Author Information
- 12) What is the APT command to add an additional software repository in Ubuntu/Debian, named: ppa:linux-libre/ppa, to your APT system?
- a. `sudo add-repository ppa:linux-libre/ppa`
 - b. `sudo add-apt-repository ppa:linux-libre/ppa`
 - c. `sudo apt-add-repository ppa:linux-libre/ppa`
 - d. `sudo apt-add ppa:linux-libre/ppa`
- 13) Which distro(s) supports the .deb package?
- a. Ubuntu only
 - b. Debian Family
 - c. Debian and RedHat
 - d. None of the above
- 14) Which distro(s) supports the RPM package?
- a. CentOS only
 - b. RedHat Family
 - c. Debian and RedHat
 - d. None of the above
- 15) We talked about using GCC to compile and install software, what was the other language/package manager discussed in the chapter?
- a. G++
 - b. APT
 - c. Python
 - d. None of the above
- 16) Describe the purpose of Virtualbox Guest Additions?
- 17) What is the RPM command to install a package from the command line?
- a. `rpm -qa *.rpm`
 - b. `rpm install *.rpm`
 - c. `rpm -q *.rpm`

- d. `rpm -i *.rpm`
- 18) After building software from source and running the `./configure` command, what is the next step?
- a. Run the `make install` command
 - b. Run the `sudo make install` command
 - c. Run the `install` command
 - d. Run the `make` command
- 19) What is the name of the new package managers developed by Canonical and RedHat?
- a. flatpak and apt
 - b. flatpak and snap
 - c. snapcraft and flatter
 - d. dnf and apt
- 20) What is the DNF command used to install additional software repositories? Use this URL to an RPM: http://download1.rpmfusion.org/free/el/updates/7/x86_64/r/rpmfusion-free-release-7-4.noarch.rpm
- a. `sudo dnf install repo http://download1.rpmfusion.org/free/el/updates/7/x86_64/r/rpmfusion-free-release-7-4.noarch.rpm`
 - b. `sudo dnf http://download1.rpmfusion.org/free/el/updates/7/x86_64/r/rpmfusion-free-release-7-4.noarch.rpm`
 - c. `sudo dnf install http://download1.rpmfusion.org/free/el/updates/7/x86_64/r/rpmfusion-free-release-7-4.noarch.rpm`
 - d. `sudo install http://download1.rpmfusion.org/free/el/updates/7/x86_64/r/rpmfusion-free-release-7-4.noarch.rpm`

17.2.2 Podcast Questions

Listen/watch the FLOSS podcast number 130 with the [VirtualBox Developers](http://twit.tv/floss/130) - <http://twit.tv/floss/130>

- ~2:35 Who is Andy Hall and Achim Hasenmuller?
- ~3:00 What is Simon Phipps relationship to the VirtualBox project?
- ~4:45 What does VirtualBox do in Andy Hall's words?
- ~6:30 What other company previously owned VirtualBox?
- ~11:20 According to Simon, what is the definition of open core?
- ~14:17 How does VirtualBox fit into Oracle's business model?
- ~16:15 As of the time of the podcast (2010) how many downloads did VirtualBox have?
- ~20:25 How does VirtualBox handle virtualized I/O?
- ~22:40 What did Intel and AMD introduce to help ease virtualization in VirtualBox?
- ~26:00 What two models of network card did VirtualBox choose to represent their virtual hardware and why?
- ~27:40 What does VirtualBox almost get native performance on?
- ~29:29 How does VirtualBox treat USB devices in Guest OSes?
- ~31:00 What are 4 virtual networking modes in VirtualBox?
- ~32:30 What is the difference between NAT and Bridged networking?
- ~39:30 What Type of hypervisor is VirtualBox?
- ~51:30 Why can't you virtualize Mac OSX on VirtualBox (as of 2014)?

17.2.3 Lab

You will need to do some research and find the download links for the Linux and BSD based distros below and install them in VirtualBox. Complete each install to disk. Assume each instance listed below is 64-bit version. Take a screen shot of each desktop after logging in.

- Debian Based
 - Ubuntu 18.04 Desktop edition
 - Lubuntu 18.04 Desktop edition
 - Trisquel Linux
 - Debian 10.x
 - PureOS 8.x
- Red Hat Based
 - Fedora 30 - Workstation edition

- Centos 7.x - minimal install
- BSD based
 - OmniOS Community Edition
- Other Linux
 - Alpine Linux
 - MX Linux
 - Arch Linux
 - Intel Clear Linux Desktop
- Network Based Install
 - openSuse Tumbleweed

17.3 Chapter 04

17.3.1 Review Questions

Linux Desktop and GUI Chapter 04 review questions

- 1) What was the original and most popular Unix “Smart Terminal?”
 - a. VIC-100
 - b. VT-100
 - c. VT-220
 - d. VC-100
- 2) What is the three letter abbreviation still in use today in modern Linux to refer to “terminal devices?”
 - a. TCY
 - b. TTY
 - c. VT-100
 - d. Virtual Terminal
- 3) Why is the GNOME terminal and Windows cmd.exe terminal emulator screens 80 by 25 by default?
 - a. The technology cannot process any larger size
 - b. The technology doesn’t need to have any larger screen size
 - c. The developers of these technologies were seeking to emulate the popular VT-100 and VT-220 terminals they used prior.
 - d. Hey it’s Windows, do they need a reason?
- 4) What is the key combo sequence you can hit to switch to a new virtual terminal in Linux?
 - a. Alt + Ctrl + Del
 - b. Alt + Ctrl + F1 - F7
 - c. Right Ctrl
 - d. Shift 5 times
- 5) What is the name of the original Unix based GUI that came out of MIT in ~1984?
 - a. W
 - b. X
 - c. Y
 - d. Z
- 6) True or False – X was originally not opensourced by MIT in 1984
- 7) For Ubuntu 18.04 what is the default desktop environment being used?
- 8) What is the name of the successor GUI compositor to X being created by the X.org foundation?
 - a. KDE
 - b. Y
 - c. Wayland Project
 - d. Jennings Project

- 9) What is the name of Ubuntu's, now deprecated, GUI compositor replacement for X? (Just the name)
- 10) What are the 3 types of Linux window managers?
- 11) There are 4 major Linux desktop environments in use today: which grouping is correct?
 - a. KDE, GNOME, CDE, LXDE
 - b. KDE, GNOME, E17, GNUMSTEP
 - c. KDE, GNOME, X, LXDE
 - d. KDE, GNOME, Xfce, LXDE
- 12) What is the name of the windowing toolkit that KDE uses?
- 13) What is the name of the windowing toolkit that GNOME uses?
- 14) What is the name of the founder of the GNOME project?
- 15) When the GNOME 3 desktop environment was released in early 2012, many people were unhappy that many changes were made. There were 3 major projects started to either preserve GNOME 2 or to modify GNOME 3 significantly – what are the names of those projects?
 - a. Enlightenment, LXDE, Xfce
 - b. Mint, Unity, Mate
 - c. Unity, Mate, Cinnamon
 - d. Cinnamon, Mint, Mate
- 16) When released in 2006, what was the main development goal of the LXDE desktop environment?
 - a. GUI desktop features
 - b. Multi-desktop paradigm
 - c. Energy saving and extremely fast
 - d. Made for high end gaming systems
- 17) Which of these statements are true in regards to Linux desktop environments? (choose all that apply)
 - a. Desktop environments have a File Manager
 - b. Desktop Environments have start or action buttons and task and notification bars
 - c. Desktop environments have a changeable but consistent look-and-feel
 - d. Desktop environments have system configuration tools and user applications built in
 - e. Desktop environments have lower memory requirements than window managers
- 18) What is the default desktop environment for Fedora 28? (Name and versions)
- 19) What is the name of the button on the upper left hand corner of the GNOME 3 desktop that you use to “launch applications?”
- 20) What is the name of the default Ubuntu desktop environment as of April 2018?

17.3.2 Podcast Questions

Please answer these questions from the Fedora Project podcast on [FLOSS - http://twit.tv/floss/71](http://twit.tv/floss/71)

- ~9:11 What is the Fedora Project?
- ~11:35 How does Redhat make money on Fedora if it is free?
- ~12:30 What is the Fedora release cycle and can businesses use this release cycle?
- ~13:30 What is the relationship between Fedora and Redhat Enterprise Linux (RHEL)?
- ~25:00 What percentage of the Fedora Project is open source?
- ~35:00 On further inspection is Fedora Project really opened source according to the Free Software Foundation?
- ~36:20 Does Fedora include proprietary NVidia drivers? Why or why not?
- ~44:30 Who is the most famous Fedora user?
- ~1:01:00. What is the difference between Centos and RHEL?

17.3.3 Lab

Using the virtual machines you installed in the previous chapter, you will now install additional software, themes, desktop environments, and window managers. You will combine screenshots of this newly installed software into a single document for submission.

Deliverable: Take screenshots of all successful install of software. Create a folder called Chapter-04 in your itmo-556 folder in your private GitHub repo. Create a file called `Readme.md` and create an H2 header for each item and place a screenshot of the result below it.

17.3.3.1 Package based install

Clone the source code repository for the text book <https://github.com/jhajek/Linux-text-book-part-1> to your Ubuntu 18.04 desktop and Fedora 28 desktop. Follow the install instructions in the repo to build a copy of the book. Take a screen shot of the title page of the PDF

17.3.3.2 GNOME Software Store

You will install assorted software using the Gnome Software application in Fedora 30. The list of software is as follows:

- 1) Inkscape
- 2) Audacity
- 3) GNOME Tweak Tool
- 4) Chromium Web Browser
- 5) Xed (text editor)

17.3.3.3 GNOME 3 Extensions

You will need to install two GNOME 3 extensions from <https://extensions.gnome.org>. The first one is the example in the book called *Caffeine*. Make sure you have the Gnome Shell integration installed to configure and install plugins directly from the browser.

- 1) Caffeine
- 2) Dash to Dock
- 3) Freon (first requires you to install the package `lm_sensors`)
- 4) Hide Dash

17.3.3.4 Snaps Install

To install and configure snaps, reference chapter 3. To show these packages are installed take a screenshot of the output of the command: `sudo snap --list`. Install these packages via Snap on Ubuntu 18.04 desktop:

- 1) Android Studio
- 2) Blender
- 3) Slack
- 4) Skype
- 5) Visual Studio Code

17.3.3.5 Flatpak Install

To install and configure flatpak and flathub, reference chapter 3. To show these packages are installed take a screenshot of the output of the command: `sudo flatpak --list`. Install these packages via flatpak on the Fedora virtual machine:

- 1) Kdenlive
- 2) Visual Studio Code
- 3) LibreOffice
- 4) AbiWord
- 5) VLC

17.3.3.6 Ubuntu 18.04 Flatpak Install

To install and configure flatpak and flathub, reference chapter 3. To show these packages are installed take a screenshot of the output of the command: `sudo flatpak --list`. Install these packages via flatpak:

- 1) Remmina
- 2) Shotcut
- 3) Fondo
- 4) VLC

17.3.3.7 AppImage Install

Find 3 [AppImage install packages](#), follow the instructions to install and run these AppImages on both Ubuntu 18.04 and Fedora 30. Take a screenshot of the command needed to run the AppImage and the corresponding first screen of that application from the [AppImage GitHub repo](#). This can be on a system of your choice.

- 1) Subsurface
- 2) Archipelago
- 3) LibrePCB

17.3.3.8 Window Managers

You will chose 1 of the window managers from the categories listed earlier in the chapter and from the Ubuntu Software Center install them. Once installed you need to log out and restart your session. In order to change the default window manager or desktop environment you need to find the Ubuntu logo icon in the upper right hand corner of the login screen.

- 1) Compositing window manager
- 2) Stacking window manager
- 3) Tiling window manager
- 4) Install Enlightenment (E17) on Ubuntu 18.04

Note: the names of packages are not always obvious so you can use search features of package managers. For example here would be how to search for the i3 package.

- `sudo apt-cache search i3`
- `sudo dnf search i3`

17.3.3.9 Desktop Environments

Install these desktops, restart your system and as you login switch your desktop environment and take a screenshot of the new environment.

- 1) Install the Xfce Desktop on Fedora 30
- 2) Install the Ubuntu Mate Desktop on Ubuntu 18.04
- 3) Install the Xfce4 Desktop on Ubuntu 18.04 (not xubuntu-desktop but just xfce4 package)

17.3.3.10 Ubuntu Theme tweaking

We will use the tutorial at NoobsLab to transform our Ubuntu 18.04 Desktop into a MacBuntu. Following the instructions here: <http://www.noobsLab.com/2018/08/macbuntu-1804-transformation-pack-ready.html>

The *Tweak tool* referred in the tutorial is the GNOME tweak tool:

Deliverable: Take screenshots of all succesful install of software. Create a file called `Readme.md` and create an H2 header for each item and place a screenshot of the result below it.

17.4 Chapter 5

17.4.1 Review Questions

- 1) What is the numeric value of a file with the permissions `rwxr-r-?` a. 777 b. 700 c. 766 d. 744

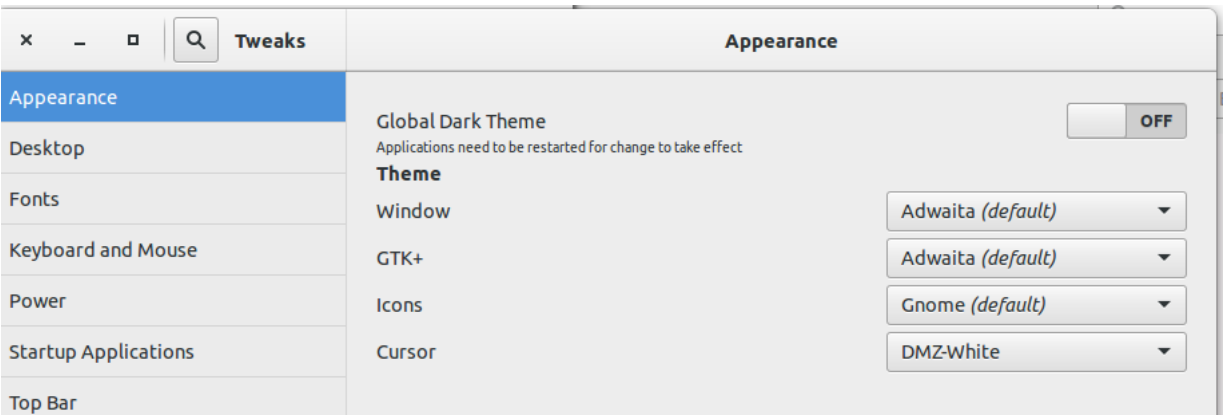


Figure 17.2: *GNOME Tweak Tool*

- 2) What is the numeric value of a file with the permissions rw—?
 - a. 700
 - b. 400
 - c. 600
 - d. 007

- 3) What is the numeric value of a file with the permission rwxr-xr-x?
 - a. 711
 - b. 755
 - c. 644
 - d. 227

- 4) What is the name of the command you use to list the contents of a directory?
 - a. lst
 - b. less
 - c. cat
 - d. ls

- 5) What is the name of the command you use to display the content of a file to the screen?
 - a. dg
 - b. tac
 - c. ls
 - d. cat

- 6) What is the name of the command you use to display the content of a file to the screen that allows you to page up and down?
 - a. more
 - b. less
 - c. ls
 - d. page

- 7) Every shell command has three components:
 - a. Command, arguments, flags
 - b. Command, options, arguments
 - c. Commands, arguments, options
 - d. Commands, flags, options

- 8) What does the user use to issue commands to the kernel?
 - a. Terminal
 - b. Commandline

- c. Magic
 - d. Shell
- 9) What is the name of the GNU shell that is standard across all Linux Distros
- a. ksh
 - b. sh
 - c. csh
 - d. bash
- 10) Based on the ls command, what is the option to do a long listing?
- a. ls -all
 - b. ls -n
 - c. list
 - d. ls -l
- 11) What is the command you can use to find out additional usage information about a shell command?
- a. about
 - b. man
 - c. F1
 - d. /?
- 12) Which of these directories is not part of the LSB LFH?
- a. bin
 - b. media
 - c. temp
 - d. opt
- 13) The Linux Filesystem is an upside down what?
- a. root
 - b. object
 - c. tree
 - d. mess
- 14) What is the name of the top of the Linux Filesystem?
- a. /
 - b. ./
 - c. ../
 - d. slashdot
- 15) Everything (directories, files, devices) in Unix/Linux is a what?
- 16) What is the name of the Unix system standard developed in ~1985 that defines at a minimum what a certified Unix based system must support?
- a. Xenix
 - b. LSB
 - c. POSIX
 - d. Linux
- 17) What is the directory where configuration files are stored in Linux?
- a. bin
 - b. sbin
 - c. etc
 - d. conf
- 18) What is the directory where all the essential command binaries are stored?
- a. bin
 - b. sbin

- c. `usr/sbin`
- d. `usr/bin`

- 19) What is the directory that holds all of the user's home directories? (no slash, just the name)
- 20) True or False - Leonart Poettering supports POSIX

17.4.2 Chapter 05 - Podcast Questions

Centos - <http://twit.tv/floss/142>

Answer said questions:

- ~2:25 Why did Randal's previous employer have a large (1000s) Red Hat system deployed?
- ~3:40 What is the short story about how Centos came about?
- ~4:58 Who is the largest commercial Enterprise Linux?
- ~7:17 How do the two projects relate to each other? How do they relate to each other from the RedHat point of view?
- ~8:10 Who from RedHat is not happy with Centos?
- ~10:25 How do Centos users differ from RedHat's paying customers?
- ~19:00. How does the RHEL environment work vs Fedora environment? (RedHat Enterprise Linux)
- ~22:55 Since Centos is using the RHEL code base how long does Centos lag behind the RHEL codebase when a new release is issued?
- ~24:24 How does Centos make money?
- ~29:00 How does Centos pay for everything?
- ~30:00 Who was Lance Davis and what happened with him?
- ~35:00 Kiran mentioned 2 million users for Centos - what did they do right compared to other projects (whitebox linux) that didn't make it?
- ~44:00 Is Centos for Servers only or can you use it on the Desktop?
- Personal questions - is what Centos doing legal? Is it ethical/moral? Why or why not?

17.4.3 Lab

The objectives of this lab is to use the shell commands we learned in this chapter and understand their proper usage patterns. The outcome will be that you will be able to successfully use the Linux Shell for navigation, file creation, and file modification. Resist the temptation to use the GUI file manager and a web browser. All actions will be done through the shell unless noted. All work can be done on either Ubuntu 18.04 desktop or Fedora 28 desktop unless noted.

Preface all screenshots with an H3 header indicated the question number: for example, Question 01, Question 02, and so forth.

- 1) Login to your Linux System. Using a package manager install the `git` program.
 - i) issue the command `git --version` and take a screenshot of the output
- 2) Navigate to your home directory and then to your Documents directory. Issue the command `git clone https://github.com/jhajek/Linux-text-book-part-1.git` (If you have done this command previously no need to redo it).
 - i) Take a screenshot of the output of the `ls` command.
- 3) Issue the `cd` command to change directory into `Linux-text-book-part-1`. Issue the command to display what type of file `.git` is. Repeat the process for the file named `./title/metadata.yaml`
 - i) Take a screen shot of the output of the previous commands.
- 4) Use the `wget` command to retrieve a copy of the Packer.io binary for Linux. Use this URL as the argument for `wget:https://releases.hashicorp.com/packer/1.3.0/packer_1.3.0_linux_amd64.zip`
 - i) Take a screenshot of the `ls` command after you have executed the previous command.
- 5) Use the `unzip` command to unzip the binary and extract the file directly to the location `/usr/local/bin`. **Hint:** use the `man unzip` command to find out the usage option in order to accomplish this.
 - i) To show this was succesful take a screenshot of the output of the command `packer -v`.
- 6) Use the `wget` command to retrieve an archived copy of the Hadoop binaries from the web. You can use this address as an argument to your `wget` command: <http://archive.apache.org/dist/hadoop/common/hadoop->

[2.7.5/hadoop-2.7.5.tar.gz](#)

- i) Take a screenshot of the `ls` command after you have executed the previous command.
- 7) Use the `tar -xvzf` command to extract the files, we will cover `tar` in later chapter. `tar` is the command `-xvzf` is the options and you need to provide the argument of the `hadoop*.tar.gz` i) Take a screenshot of the output of the `ls` command to show the extracted hadoop directory.
- 8) Find the command that is used to rename a file or directory and rename the Hadoop directory to be `ust hadoop`
 - i) Take a screenshot of the output of the `ls` command.
- 9) What would be the command to delete the file `hadoop*.zip`?
 - i) Issue the command `ls`, then type the command to remove the `.tar.gz` file, the type the `ls` command again to show it has been remove. Take a screenshot of the combined output of all 3 directories.
- 10) Using a Web Browser from your GUI, navigate to and open the file named `chapter-05-file-path-exercise.html` located under the directory `./Linux-text-book-part-1-master/files/Chapter-05/images/`: Note that the images on the web page are broken. Take a look at the source code (view source) and move the `stallman.jpg` to the proper directory to make the webpage render that image properly in a single command using the command line.
 - i) Take a screenshot of that single command and a screenshot of the web browser showing the `stallman.jpg` image properly rendering.
- 11) Using the commandline again, move the `ms-loves-linux.png` image to the proper directory using the `mv` command in a single command:
 - i) Take a screenshot of that single command and a screenshot of the output of the web browser showing both the `stallman.jpg` and the `ms-loves-linux.png` images properly rendering.
- 12) Assume your PWD is your Home directory: In a series of commands, `cd` to Documents, then create a directory named `packer-scripts`. Under this directory create 5 sub-directories, `ubuntu16-04`, `ubuntu18-04`, `fedora28`, `centos7`, `OmniOS`.
 - i) With your PWD as `~/Documents/packer-scripts`, execute the command that will give a long listing of the contents of the directory. Take a screenshot of this command as well as the output.

17.5 Chapter 06 - Review Questions

Shell Meta-Characters, Pipes, Search, and Tools

Chapter 06 review questions

1. What is the name for characters that have special meanings in the Linux shell?
 - a. special chars
 - b. marked characters
 - c. shell characters
 - d. shell meta-characters
2. Assume your `pwd` is `~`. If you wanted to list every directory only that started with the letters “Do” what would be the command?
 - a. `ls -l`
 - b. `ls -la`
 - c. `ls -l Do`
 - d. `ls -l Do*`
3. In figure 6.2 in Chapter 06 which of the 3 blue boxes is the step where shell meta-characters are transformed into text?
 - a. Lexical analysis and parse
 - b. Execution
 - c. Builtins
 - d. Expansion

4. Which meta-character allows you to string commands together regardless of the successful execution of the previous command?
 - a. &&
 - b. ;
 - c. \+
 - d. ||
5. Which meta-character allows you to string commands together but will exit if one of the commands fails?
 - a. &&
 - b. ;
 - c. \+
 - d. ||
6. Which meta-character is the wildcard (0 or more matches)?
 - a. ?
 - b. **
 - c. &
 - d. *
7. Which meta-character is the single character wildcard?
 - a. ?
 - b. '
 - c. &
 - d. *
8. Square braces [] indicate sets or _____ of characters to be processed
 - a. numbers
 - b. letters
 - c. characters
 - d. ranges
9. If you wanted to use brace expansion and create a series of nine files named: file1, file2, file3, etc etc all at once—what command would you use? (type the full command using touch .
 - a. touch file{1..9}
10. If you wanted to assign the value of /etc/alternatives/java to a shell variable named JAVA_HOME—what would be the proper syntax?
 - a. JAVA_HOME = /etc/alternatives/java
 - b. /etc/alternatives/java = JAVA_HOME
 - c. JAVA_HOME=/etc/alternatives/java
 - d. \$JAVA_HOME=/etc/alternatives/java
11. What would be the proper syntax to display the content of a variable named JAVA_HOME in the shell?
 - a. echo JAVA_HOME
 - b. cat JAVA_HOME
 - c. print JAVA_HOME
 - d. echo \$JAVA_HOME
12. There are 3 standard I/O devices in a Linux system, standard in, standard out, and _____
 - a. standard air
 - b. standard I/O
 - c. standard x
 - d. standard error
13. Standard In is what device by default?
 - a. mouse

- b. screen
 - c. tty
 - d. keyboard
14. Standard Out is what device by default?
- a. mouse
 - b. screen
 - c. X
 - d. keyboard
15. What meta-character can you use to redirect standard out to a file? (Choose all that apply.)
- a. >
 - b. »
 - c. <
 - d. »>
16. What meta-character is used to redirect the standard output of one command as the standard input of another command?
- a. ->
 - b. &&
 - c. ||
 - d. |
17. Which command is a shortcut to display the kernel's output messages?
- a. kern
 - b. &kern
 - c. top
 - d. dmesg
18. Which command is used to search within files using textual filter patterns?
- a. find
 - b. locate
 - c. slocate
 - d. grep
19. When you combine a tar (tape archive) and an additional compression method-what is the name for the resulting file?
- a. tar
 - b. tarx
 - c. tarall
 - d. tarball
20. What is the name of the GNU compression tool project released in 1992?
- a. xz
 - b. zip
 - c. DEFLATE
 - d. gzip

17.5.1 Podcast Questions

Listen to the FLOSS podcast number 73 with [Tim O'Reilly](http://twit.tv/floss/73) - <http://twit.tv/floss/73>

- Who is Tim Orielly? ~3:00-5:00
- What is Oscon? ~6:45
- Who coined the term web 2.0? ~13:34
- What did we learn from the IBM PC? ~18:30
- What is web 2.0? ~19:30
- Open Source vs Open Data - what does Tim Orielly think is the ultimate destination for computing? ~23:00

- Where is the money made in open source - software or data? ~ 34:00
- What prediction did Tim O'Reilly make in this podcast (2009) that is now coming true? ~51:32
- radar.oreilly.com What is the lag time from articles on this site to the main stream media? ~55:00

17.5.2 Lab - Chapter 06

The objectives of this lab will be to use the shell and understand meta-characters, pipes, search, and tools. The outcome will be that you will be able to successfully use meta-characters for file creation, location, modification, and manipulation. You will successfully master the concept of pipes and redirection as well. Resist the temptation to use the GUI file manager and a web browser. All actions will be done through the shell. You can use either an Ubuntu or a Fedora based OS. **Deliverable:** Take a screenshot demonstrating the required output of each command.

1. Issue the command to clone a copy of the textbook code (if you have already done this in Lab 5 no need to repeat the step). Issue the command to `cd` into the `Linux-Text-Book-Part-I` directory. Type the command that will list every file in this directory that has any number of characters and a `.sh` two character file extension of any name
2. Type the command inside the textbook directory will do a *long listing* of Chapters-02, 04, 06, and 08 only
3. Type the command that will copy the file `Chapter-02/chapter-02.md` into the your home directory, then list the content of your home directory. Use the meta-character needed to execute the proceeding commands only if the previous command is true and place all these commands into one single line
4. In your home directory, using the meta-character, create these two series of files: `ubuntu10.txt - ubuntu15.txt` and `redhat10.txt - redhat15.txt`. Create each series using a single command
5. In your home directory, using the meta-character, issue a command to list only the `ubuntu10.txt - ubuntu15.txt` files
6. In your home directory, using the meta-character, create the directories named: `debian10`, `debian11`, and `debian12` with a single command
7. In your home directory, using the meta-character, delete the directories named `debian10`, `debian11`, and `debian12` with a single command
8. In your home directory redirect the output of the `date` command into a file named: **today.txt**
9. In your home directory append the output of the `date +%m%d%Y` command to the file **today.txt** and display the content - you should see two formatted date entries
10. Create a shell variable named `UT`, assign the contents of the command `uptime` to `UT` and print a string to the screen with its value and with a string stating, "The system has been up for:" and then the value of `UT`.
11. Using an Ubuntu Desktop system, execute the following commands: `sudo apt-get -y update 1>/tmp/01.out 2>/tmp/01.err sudo apt-get -y install nginx 1>/tmp/02.out 2>/tmp/02.err` and `sudo systemctl start nginx 1>/tmp/03.out 2>/tmp/03.err`. Issue the command to list the contents of the `/tmp` directory.
12. Display the contents of the `*.out` files using a meta-character in one command and pipe the output to the `less` command
13. Display the contents of the `*.err` files using a meta-character in one command and pipe its output to the `less` command.
14. Type the command `ls -l /topsecret` and redirect both standard out and standard error to a file named: `/tmp/out-and-error.txt`
15. You will find a file named `hosts.deny` located in the directory `files > Chapter-06` of the download of the textbook. It contains a list of IP addresses - what command would you use to count ONLY the number of lines in the file?
16. Using the `error.log` file located in the directory `files > Chapter-06` - what command would you use to count only unique lines and to display their count and only if there is more than 1 occurrence?
17. What command would let you display the content of the `hosts.deny` file, cut out the the second column and sort it?
18. What command would let you search the file `error.log` for the lines that contain the term **robots.txt**?
19. What command would let you count the number of lines that have the term "robots.txt" in the file `error.log`?
20. Using the `hosts.deny` file, what command would you type to display the last 10 lines of the file, cut out the field with the IP address and sort them in ascending order?
21. Use the `grep` tool to search the file `error.log` for the line "Invalid method in request" and print to the screen the lines found.
22. Using tools discussed in this chapter and based on the contents of the file **error.log**, type the commands needed to find the following information: How many unique IPs did the error messages come from?

23. How many unique URLs based errors (last column), and list all of the unique type of errors (second to last column).
24. Using the `find` command and starting from the `~` directory what would be the command to find all files with the name `.md`?
25. Using the `find` command and starting from your home directory, what would be the command to find all the files that have been modified in the previous hour?

17.6 Chapter 07

17.6.1 Review Questions

Introduction to Linux Editors, Introduction to Shell Scripts, and User Profiles

Chapter 07 review questions

1. What are the two main representatives of stream editors?
 - a. gedit and kate
 - b. Nano and Joe
 - c. vi and Nano
 - d. vi and Emacs
2. Which family of editors came first?
 - a. Screen editors
 - b. Butterfly editors
 - c. GUI editors
 - d. Stream Editors
3. Emacs at its core is what?
 - a. A fine operating system in need of a good editor
 - b. The heart and soul of hackers
 - c. An interpreter for Emacs Lisp
 - d. A C program
4. Who created the vi editor?
 - a. Richard Stallman
 - b. Doug McIlroy
 - c. Bill Joy
 - d. Dave McKenzie
4. What year was vi released in?
 - a. 1972
 - b. 1979
 - c. 1999
 - d. 1978
5. Which of the following sequences of the history of vi is correct?
 - a. Emacs -> ed -> ex -> vi
 - b. ed -> em -> ex -> vi -> vim
 - c. em -> ex -> vi
 - d. em -> ed -> vi -> vim
6. What are the three modes in vi?
7. What is the key you use in vi to transition between COMMAND MODE and INSERT mode?
8. What command sequence (key) in vi will add text to the right of the current cursor position? (just the letter)

9. What command sequence (key) in vi will move you to the beginning of the next word? (just the letter)
10. What command sequence in vi will delete a single line based on the current cursor position? (just the letters)
11. What command sequence in vi will delete 10 lines from the current cursor position? (just the numbers and letters)
12. Which command in ex mode (vi) will save the current file you are working on and exit the vi editor? (include the “:”)
13. In the log file u_ex150911.log what would be the ex command to search forward for occurrences of YandexBot? (include the forward slash)
14. Assuming your pwd is Linux-text-book-part-I and you have loaded Chapter-02.chapter-02.md into vi, what would be the ex mode command to replace all occurrences of linux with Linux?
15. Assuming your pwd is Linux-text-book-part-I and you have loaded Chapter-02.chapter-02.md into vi, what would be the ex mode command to replace all occurrences of Linux with GNU/Linux? (remember to escape the /)
16. Assuming the your pwd is Linux-text-book-part-I and you have loaded Chapter-02.chapter-02.md into vi, what would be the ex mode command to remove all occurrences of the word Windows?
17. Assuming a file name topsecret.sh has a permission of 644 - what is the shortcut to give the owner of the file permission to execute the script?
18. Assuming a file named moretopsecret.sh has a permission of 757 - what is the shortcut to remove all permissions from the the other group?
19. What is the correct sequence of profile inheritance?
 - a. /etc/profile -> ~/.bash_profile or ~/.bash_login or ~/.profile -> ~/.bashrc
 - b. ~/.bashrc -> ~/.bash_profile or ~/.bash_login or ~/.profile -> /etc/profile
20. What is the command to display the contents of the PATH system variable on the command line?
 - a. echo PATH
 - b. echo \$PATH
 - c. echo path
 - d. \$PATH

17.6.2 Podcast Questions

Listen to the FLOSS podcast number 88 with [Linus Torvalds - http://twit.tv/show/floss-weekly/88](http://twit.tv/show/floss-weekly/88)

- ~6:32 Who is Linus Torvalds?
- ~6:54 Where did he create Linux?
- ~7:30 What did Unix have that other operating systems didn't at that time?
- ~10:02 Within a few months of Linux first release roughly how many people were interested in Linux?
- ~10:30 About what month and what year did this happen?
- ~10:40-13:30 What was the initial inspiration to create the Linux Kernel as an open source project?
- ~13:30-14:00 Why was it licensed under the GPL?
- ~20:48 Why didn't Linus want to work for a Linux company?
- ~41:00 More than the technology hurdle what else is needed to get into Linux Kernel Development?
- ~46:10 What is the way to become a great programmer?
- ~51:17 What is Linus' farewell message to the audience?

17.6.3 Lab Chapter 7

Objectives: The objective of this lab is to master vi commands and shell scripts

Outcomes: At the end you will have mastered the basics of vi and now be proficient in all the tools of Linux shell scripting

Prereqs: You will need to install the program `vimtutor` for the first part. You can do that on Ubuntu by typing `sudo apt-get install vim vim-runtime vim-gtk` and on Fedora by typing `sudo dnf install vim vim-enhanced`.

- 1) To begin type the command `vimtutor` from the commandline. **Warning:** `vimtutor` requires you to read the instructions carefully.
 - i) This is a 6 part tutorial. You need to follow all the steps of the 6 part tutorial making your changes directly in the file.
 - ii) **Be careful** to save the file to an external location – otherwise IT WILL BE OVERWRITTEN each time you launch the `vimtutor` command. You can do this by typing `:w ~/Documents/vimtutor.txt` - this way you can edit the file on your local system instead of launching the `vimtutor` application again. Note you need to use `vim` for this assignment.
- 2) Inside of the `files/Chapter-07/lab` folder using `vim` open `install-java-8-openjdk.sh`. You will be using `vim` to modify this file
 - i) Use the `ex` command to find all occurrences of `;` and replace them with `&&`.
 - ii) Using `vim` commands replace all occurrences of the numbers `2.6.5` with `2.8.5`.
 - iii) Using `vim` to append the packages `python` and `python-dev` to line 18.
 - iv) In the shell script, `install-java-8-openjdk.sh`, using `vim` insert a comment above each line explaining what the command is doing.
- 3) In `vim` using create a shell script named `created-shell-script.sh` to run in Ubuntu with the following requirements, you will need to reference chapter 03=07 as well: (resist writing it in notepad)
 - i) Create a script with the proper shebang on the first line.
 - ii) Type the command to update package repositories.
 - iii) type the command to install the `java 8 openjdk` and the `software-properties-common` package with the flag to auto accept any question.
 - iv) Type and chain the commands to use `wget` and retrieve this tarball: <http://archive.apache.org/dist/hadoop/common/hadoop-2.8.5/hadoop-2.8.5.tar.gz> then extract it—in one line.
 - v) Type the command to install these packages with the auto accept flag turned on: `pkgconf wget liblz2-dev sysstat iotop vim libssl-dev libsnappy-dev libsnappy-java libbz2-dev libgcrypt11-dev zlib1g-dev lzop htop fail2ban`
 - vi) Type the command to upgrade the Ubuntu distribution and redirect the standard output to `/tmp/distupgrade.out`
 - vii) Create a shell variable named `RESULT`, set the result of the command `sudo find / -name tools.jar` to this value and run the command to display the content of the `$RESULT` variable.
 - viii) Add these lines to the END of your shell script:

```
cat << EOT >> ~/.bashrc
```

```
##### Inserted by User #####
export JAVA_HOME=/usr
export HADOOP_HOME=$HOME/hadoop-2.8.5
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
export HADOOP_CLASSPATH=/usr/lib/jvm/java-8-openjdk-amd64/lib/tools.jar
EOT
```

Source the `.bashrc` file with the command `. ~/.bashrc`, execute the command `hadoop version` if version information outputs successfully then you have succeeded.

Deliverable:

Submit your Github URL for your repo to Blackboard.

- 1) Push the file `install-java-8-openjdk.sh` in your week-07 > itmo-556 Github repo.
- 2) Push the file `vimtutor.txt` to your week-07 > itmo-556 Github repo.
- 3) Push the file `created-shell-script.sh` to your week-07 > itmo-556 Github repo.

17.7 Chapter 08

17.7.1 Review Questions

- 1) True or False The Bash shell scripting language has traditional language constructs like C or Java?
- 2) What meta-character do you use to access the content of a shell variable?
 - a. \$
 - b. #
 - c. !
 - d. No character - trick questions
- 3) When assigning the standard output of a command to a variable what characters do you encase the command in?
 - a. “
 - b. \$
 - c. ""
 - d. No characters - trick questions
- 4) True or False - You can include shell meta-characters inside of two backticks `` - **example:**ANS=`ls -l test[1-5]`
 - a. DIR='ls -l ./test[1-4].txt'
 - b. "DIR = `ls -l ./test[1-4].txt`"
 - c. "\$DIR=`ls -l ./test[1-4].txt`"
 - d. "DIR=`ls -l ./test[1-4].txt`"
- 5) Which command will list the names of any file that matches these names: file1.txt file2.txt file3.txt file4.txt and send the content of that output to a variable named DIR?
 - a. DIR='ls -l ./test[1-4].txt'
 - b. "DIR = `ls -l ./test[1-4].txt`"
 - c. "\$DIR=`ls -l ./test[1-4].txt`"
 - d. "DIR=`ls -l ./test[1-4].txt`"
- 6) Which of these are valid commands in the first line of a shell script? (Choose any - assume any paths are valid paths to executables)
 - a. #!/bin/bash
 - b. !#/bin/bash
 - c. #!/usr/local/bin/bash
 - d. #/bin/bash
 - e. #!/bin/ksh
- 7) If you stored the output of the command hostname into a variable named sys-hostname, what would be the command to print the content to the screen?
 - a. echo \$HOSTNAME
 - b. echo \$hostname
 - c. echo \$SYS-HOSTNAME
 - d. echo \$sys-hostname
- 8) What is the name of the command to print out all the predefined system variables?
- 9) What is the name of the command that allows you to take stdout of a command and insert the lines of output into an array?
 - a. arrayfile
 - b. declare
 - c. for loop
 - d. mapfile
- 10) Which of these is a valid command to take the output of this find command and assign the contents to an array? (Assume the array name has already been declared. Choose one)
 - a. mapfile SEARCHARRAY = (find ~ -name mozilla*)
 - b. mapfile SEARCHARRAY < < (find ~ -name mozilla*)
 - c. mapfile -t SEARCHARRAY < <(find ~ -name mozilla*)

- d. `mapfile -t SEARCHARRAY < (find ~ -name mozilla*)`
- 11) Which below is a valid command to find the LENGTH of an array?
- `${#SEARCHARRAY[@]}`
 - `${SEARCHARRAY[@]}`
 - `${SEARACHARRAY[#]}`
 - `${@SEARCHARRAY[#]}`
- 12) Based on this shell script and positional parameters, what would the command be to print out the first positional parameter after the script name? `./delete-directory.sh ~/Documents/text-book Jeremy` a. `echo $0` b. `echo $1` c. `echo args[1]` d. `echo ${1}`
- 13) Based on this shell script and positional parameters, what would the command be to print out the entire content of the positional parameter array? `./delete-directory.sh ~/Documents/text-book Jeremy`
- `echo $#`
 - `echo @!`
 - `echo $0`
 - `echo $@`
- 14) Based on this shell script and positional parameters, what would the command be to print out the LENGTH of the positional parameter array? `./delete-directory.sh ~/Documents/text-book Jeremy`
- `echo $#`
 - `echo @!`
 - `echo $0`
 - `echo $@`
- 15) In a Bash IF statement, what is the name for the pre-made test conditions?
- Primaries
 - Secondary expressions
 - Expression
 - Primary expressions
- 16) All values in a Bash IF statement are of what data type by default?
- INT
 - STRING
 - NULL
 - CHAR
- 17) Which of these answers will execute a shell script named `~/backup.sh` at 2 am every night of the week?
- `* * * * * ~/backup.sh`
 - `2 * * * * ~/backup.sh`
 - `* 2 * * * ~/backup.sh`
 - `* * * 2 * ~/backup.sh`
- 18) Which of these answers will execute a shell script named `~/clean-directory.sh` every 15 minutes?
- `3/15 * * * * ~/clean-directory.sh`
 - `*/15 * * * * ~/clean-directory.sh`
 - `* 3/15 * * * ~/clean-directory.sh`
 - `* */15 * * * ~/clean-directory.sh`
- 19) Which of the crontab builtins would you use to execute a cron job 1 time a year on midnight of January 1st? The name of the script is `~/give-free-cash-to-students.sh`
- `* * * * 1 ~/give-free-cash-to-students.sh`
 - `1 * * * * ~/give-free-cash-to-students.sh`
 - `1 1 1 1 1 ~/give-free-cash-to-students.sh`
 - `@yearly ~/give-free-cash-to-students.sh`
- 20) What is the name of the control structure that allows you to incrementally through the contents of an array?

- a. IF
- b. CASE
- c. UNTIL
- d. FOR

17.7.2 Podcast Questions

Command Line Heroes: Bash

- ~0:20 Who is the creator of the Bash Shell?
- ~0:43 Which organization did the creator of the Bash Shell write the shell for?
- ~2:05 How does the podcast host define a shell script?
- ~2:23 Shell scripts are the key to what?
- ~3:28 When did Ken Thompson release his shell and what was it missing?
- ~3:45 What year was shell scripting come into existence?
- ~4:27 What was the shell that became the AT&T UNIX standard shell?
- ~5:53 The Bourne Shell was licensed and owned by whom?
- ~7:59 Why was Brian Fox the perfect person to develop the Bash Shell?
- ~9:30 How long did it take to create the Bash shell and what was difficult about this?
- ~12:02 What did Brian accidentally do to the Bash Shell?
- ~14:48 What was the other shell released one month before Bash?
- ~15:19 When was GNU Bash released?
- ~18:40 What was the released/intended purpose of GNU Bash?
- ~19:25 What words and terms are in now in use in everyday English?
- ~20:46 Was Steven Bourne “cool” with the Bash Shell?
- ~22:52 What prepares you to be more of a long-term thinker?

17.7.3 Lab

17.7.3.1 Lab Objectives

This lab will allow you to create shell scripts. Use positional parameters, control structures, and write cron jobs.

17.7.3.2 Lab Outcomes

At the completion of this lab you will further your knowledge of shell scripting and enhance your abilities using Bash shell scripts.

In the GitHub repo provided to you please create a folder in your ITMO-556 directory named Chapter-08. In this directory you will create a file called ReadMe.md and all of the answers, screenshots, and code will be contained in this document. Submit to Blackboard just your GitHub URL.

- 1) What would be the command to create an array in Bash named itemARRAY?
- 2) Write a shell script that declares an array in Bash named `dirarr`. Using the `mapfile` command - redirect the output of the `ls -l ~` command into the array previously named and echo out the 3rd and 4th elements of that array.
- 3) Write a WHILE loop that will read the content of the file `names.txt`, (located in the files > Chapter-08 > lab folder) and create a directory based on the value on the line in the `/tmp` directory (one per users). Include an if statement to detect if the directory already exists, if it does exist, write the duplicate name out to a text file named: `duplicates.txt` located in the `/tmp` directory.
- 4) Write the syntax to make a cronjob execute 5 minutes past every hour everyday to execute the shellscript you previously made to store the content of `ls -l ~` into an array named `dirarr`.
- 5) Locate the file `install-java8.sh` located in the files > Chapter-08 > lab directory. Modify the script to include IF statements to check for the existence of the path `/datapool1` and to print an error message if the path does not exist.

- 6) Modify `install-java8.sh` again—this time take a positional parameter and put that in place of the directory name `/datapool1` (this will allow you to customize the install location of the shell script).
- 7) Modify the `install-java8.sh` from the previous question to count the number of positional parameters and if less than 1 or more than 1 stop execution of the script (`exit`).
- 8) Create a directory in `~` named `topsecret`. In that directory create a file named `xfile.txt`. Write a shell script to check if that file has executable permission by passing the filename as a positional parameter. If `TRUE` print a message. If `FALSE` print an error message that the positional parameter name of the file is not executable.
- 9) Write a shell script to check in the `~/topsecret` directory to see if a given file name exists (passed in by positional parameters). If `TRUE` print a message else print an error message with the given file name being passed.
- 10) Write a shell script to check if a given `PATH` is a file or a directory. If `TRUE` print a message, else print an error message using the given file name.
- 11) Write a shell script that takes 4 positional parameters. In the shell script print out `$0`, `$#`, and `$@` with an explanation of what these variables contain.
- 12) Repeat the previous cron command but this time redirect the standard out and standard error to a file named `~/Documents/my.log`
- 13) Using `awk` and other tools, how would you find which ip caused the most HTTP 404 errors? Take a screenshot of the command and the output. Use these two files in `files/Chapter-08/logs`: `u_ex150721.log`, `u_ex151002.log`.
- 14) Using `awk` and other tools, how would you capture the top 5 offending IPs? Take a screenshot of the command and the output. Use these two files in `files/Chapter-08/logs`: `u_ex150721.log`, `u_ex151002.log`.
- 15) Using `sed`, type the command to find the line **bind-address** located in the mariadb database server config file (you might need to install `mariadb-server`). The file locations are: Fedora `/etc/my.cnf.d/mariadb-server.cnf` and Ubuntu `/etc/mysql/mariadb.conf.d/50-server.cnf`. Comment out the value, change the IP value to `0.0.0.0`, and write the change back to the original file. Take a screenshot of the output.

17.8 Chapter 09

17.8.1 Review Questions

- 1) What user account has superuser privilege in Linux?
 - a. `sudo`
 - b. `su`
 - c. `superuser`
 - d. `root`
- 2) Which command do you use to temporarily elevate your user's privilege to the superuser (`root`)?
 - a. `su`
 - b. `sudo`
 - c. `su -`
 - d. `root`
- 3) How can you display the content of a file named `topsecret.txt` that has permissions `000` and is owned by another user?
 - a. You can't do that
 - b. `root cat topsecret.txt`
 - c. `sudo cat topsecret.txt`
 - d. `su cat topsecret.txt`
- 4) What license is the `sudo` application under?
 - a. GPL
 - b. BSD
 - c. Public Domain

- d. ISC
- 5) Which operating system doesn't have an active root account by default?
 - a. Debian
 - b. Ubuntu
 - c. All Debian based distros
 - d. Fedora
- 6) What is the name of the file where sudo privilege are kept?
 - a. /etc/sudo
 - b. visudo
 - c. /etc/allow
 - d. /etc/sudoers
- 7) What is the name of the command used to modify /etc/sudoers to grant a new user sudo privilege?
 - a. Just use vi to edit it directly
 - b. Logout and log back in as root and do it
 - c. visudo
 - d. sudo visudo
- 8) Based on this line in /etc/sudoers - `%meninblack ALL=(ALL:ALL) ALL` - what does the first value by the % mean?
 - a. Name of a group
 - b. Name of a user
 - c. Name of the user group
 - d. Name of a process
- 9) In the /etc/sudoers file - what does this line mean: `RMS ALL=(root) NOPASSWD: ALL`
 - a. The user RMS has sudo permissions and access to all commands
 - b. The user RMS has sudo permissions
 - c. The group RMS has sudo permissions to all commands
 - d. The user RMS has sudo permissions and access to all commands, and requires no password to elevate to the sudo user
- 10) When using the su command to switch from a regular user account to the root user account, what do you type to return to the standard user account?
 - a. quit
 - b. exit
 - c. stop
 - d. sudo reboot
- 11) What command would you use to edit the file at this location: /var/www/html/index.html?
 - a. vi /var/www/html/index.html
 - b. sudo vim /var/www/html/index.html
 - c. vim /var/www/html/index.html
 - d. You need to **chown** the file and change the owner
- 12) On a Linux system, which directory are all the traditional system (non-systemd) logs kept in?
 - a. /var/run
 - b. /logs
 - c. /var/adm/log
 - d. /var/log
- 13) Under systemd and journald where are the logs kept?
 - a. /var/log
 - b. /var/log/error
 - c. /var/log/journald

- d. Trick question - as logs are stored in a binary format and retrieved via journalctl
- 14) What is the command you use to query the system logs in systemd?
- systemctl
 - journald
 - journalctl
 - showlogs
- 15) How would you filter the systemd log based on time? (Which is valid syntax?)
- journalctl --since=yesterday
 - journalctl --since=tomorrow
 - journalctl --yesterday
 - journalctl --filter=yesterday
- 16) Where is the journald.conf file located?
- /etc/logrotate.conf
 - /etc/systemd/journalctl.conf
 - /etc/systemd.conf
 - /etc/systemd/journald.conf
- 17) What command provides a dynamic real-time view of a running system?
- top
 - iostat
 - ranwhen
 - journalctl
- 18) Debian based distros have an additional command to abstract the process to add users to the system - what is it?
- useradd
 - usermod
 - adduser
 - add
- 19) What command would be used to modify a user account settings and add them to the sudo users group on an Ub untu distro (user is named controller)?
- sudo useradd -aG sudo controller
 - sudo usermod -aG sudo controller
 - sudo usermod -G sudo controller
 - sudo userdel controller
- 20) Which below are valid useradd commands? (Choose all that apply)
- sudo useradd -c "User for spring class" -d "/home/export/controller" -G sudo -s /bin/ksh -m controller
 - sudo useradd -D controller
 - sudo useradd controller
 - sudo useradd -G sudo -s /bin/ksh -m controller
 - sudo useradd -c "User for spring class" -G sudo -m controller

17.8.2 Podcast Questions

NodeJS - <https://twit.tv/shows/floss-weekly/episodes/387>

Node.js Update

- ~4:20 Where does Aaron (guy wearing black) say he is starting to see Node JS more and more?
- ~7:22 What is Node.js?
- ~8:50 According to Mikael, what are we seeing an explosion of?
- ~9:15 What language(s) is NodeJS similar to?

- ~9:35 What is Event Driven Programming?
- ~10:45 what is NodeJS package manager/ecosystem?
- ~12:05 what kind of things would use Node?
- ~17:20 What are the two popular desktop apps built in NodeJS that Mikael mentioned?
- ~ 19:30 What are some of the ways to learn NodeJS?
- ~ 21:30 What did Microsoft do with NodeJS and where did it get Node?
- ~ 22:30 Mikael mentions multiple languages: TypeScript, CoffeeScript, Electron, and Dart – what are they and how do they relate to the NodeJS project? (Need to do some side research)
- ~25:27 What company started the NodeJS and eventually the NodeJS foundation? (Currently owned by Samsung)
- ~ 26:43 What does the NodeJS foundation do (what is its role?)
- ~ 34:00 Who is the guest and what is his job?
- ~ 38:48 How much did Mikael reduce his code when he switched from Python to NodeJS?
- ~41:54 is it possible to run NodeJS for command line scripting?
- ~45:00 Where is pretty much every NPM module hosted?
- ~47:40 What is the Go language good at and what is it not?
- ~50:55 Is there any relationship between NodeJS and Docker?

17.8.3 Lab

Objectives

The objective of this lab are as follows:

- Understand when and how to use the sudo command
- Understand how to edit the `/etc/sudoers` file
- Understand how to use the journald and journalctl logging mechanism in systemd
- Understand how to add and manage user accounts
- Understand the structure and use of the cron service
- Understand how to modify, use, and secure the SSH service

Outcomes

At the outcome of this lab you will be able to successfully understand how to apply the sudo/root user paradigm. You will understand the binary logging mechanism of journald. You will be able to add, delete, and modify user accounts. Finally you will be able to schedule shell scripts to execute at repeated intervals.

Note Submit the answers all contained in one shellscript with a comment above stating which question the code answers **Note** If a command asks you to work on a user that doesn't exist it is assumed that you have to create it. **Note** The `mysqldump` application requires the `mysql-client` package to be installed. <http://superuser.com/question/s/165582/installation-mysqldump>

- 1) What would be the command to add a user named “controller” to your system - using the system default values?
- 2) What would be the command to modify the user's group to add them to a *superuser* group (sudo on Ubuntu or wheel on Fedora based)?
- 3) What would be the command to delete a user account named nsa-spy? (Note you also have to include the steps to add this user... unless the NSA is already in your system =)
- 4) What would be the command to edit the `/etc/sudoers` file and give the user “mysql-backup” sudo privilege? (Show the `/etc/sudoers` being edited and enter the relevant line that you add to that file)
- 5) What would be the command to edit the `/etc/sudoers` file and give the group “mysql-admins” sudo privilege? (Show the `/etc/sudoers` being edited and enter the relevant line that you add to that file)
- 6) What would be the command to edit the `/etc/sudoers` file to give the user “mysql-admin” sudo privilege to only use the mysql database backup command “mysqldump” ? (Show the `/etc/sudoers` being edited and enter the relevant line that you add to that file)
- 7) What would be the command to edit the `/etc/sudoers` file to give the user “mysql-admin” sudo privilege to only execute the `mysql` command and not require a password?

- 8) When you execute the command `tail journalctl` - you receive an error? Show the error in a screenshot and explain why the error comes?
- 9) What would be the command to execute to find all the occurrences of logs generated by SSHD in journalctl? P.168 in the text book – you may need to install `openssh-server` package if the command returns no results
- 10) What would be the command to execute to find all the logs generated by `__PID=1` (systemd itself) and since yesterday?
- 11) What would be the command to execute to see the logs of the current boot only using journalctl?
- 12) Which file and what value would I modify to change the journal's settings to make the logs be stored in memory (volatile)?
- 13) The `journal` values `SystemMaxUse=` and `RuntimeMaxUse=` default to 10% and 15% of the system disk respectively. How would you modify that value to be 20% and 30% respectively? (Note you can't add percentages, you have to use your system and do some scratch math - you can execute a `df -H` command to see the size of your root partition)
- 14) What would be the command to edit the cron service and run this command “`mysqldump -xml -u root world City`” (Assume you have mysql installed) at 2 am every Sunday?
- 15) What would be the command to edit the cron service and run this command “`mysqldump -xml -u root world City`” (Assume you have mysql installed) at the 1st day of every month?
- 16) What would be the command to edit the cron service and run this command “`mysqldump -xml -u root world City`” (Assume you have mysql installed) every 45 minutes?
- 17) What would be the command to edit the cron service and run this command “`mysqldump -xml -u root world City`” (Assume you have mysql installed) on every 45 minutes past the hour on Sundays?
- 18) What would be the command to edit the cron service and run this command “`mysqldump -xml -u root world City`” (Assume you have mysql installed) at 2:45am on this coming Tuesday only?
- 19) What command would you use to change the group ownership of the file `todo-list.txt` to be owned by the “accounting” group? (If that group doesn't exist then create it on your system)
- 20) What would be the command you would type to generate a RSA key pair?
- 21) What would be the command to transfer an RSA key pair to a remote system named `logserver` with the username `worker`?
- 22) What would be the command to connect via ssh using the identity `logserver` and connect to the non-standard port of 5555 instead of the default 22?
- 23) The next questions require some setup:
 - i. You need two virtual machines for this part: One Ubuntu based and one Fedora based (or two comparably different OSes, FreeBSD, Trisquel, etc etc).
 - ii. You will need to modify the Network settings to **Bridged** in Virtualbox to get a public IP (if you are at home your router should suffice, if you are on campus you can come to the lab).
 - iii. Install `openssh-server` on Fedora.
 - iv. Clone the repository <https://github.com/arthepsy/ssh-audit> to both the client and server system
 - v. Run the ssh audit on the client and server, list the weak ciphers installed by default
- 24) Modify the client and servers using the example in the text to increase cipher strength, run the `ssh-audit` tool again and report any weak ciphers or security anomalies.
- 25) On the SSH server make the following changes to the `sshd_config` file and paste them at the end of the `ReadMe.md` file
 - i. Not accept any password based authentication attempts
 - ii. Change the default port to be 5555
 - iii. Disable the value `PermitRootLogin`

17.9 Chapter 10

17.9.1 Review Questions

- 1) What is the name of *beep* sound heard in the initial boot of a PC (assume you are using BIOS not UEFI)?
 - a) PERC
 - b) POST
 - c) GRUB
 - d) BIOS
- 2) What is the name of the GNU software that is the first software program that runs when a computer with Linux installed starts?
 - a) BIOS
 - b) LILO
 - c) GRUB
 - d) GLOADER
- 3) In what Linux directory is the kernel and initrd image stored?
 - a) /root
 - b) /root/kernel
 - c) /boot
 - d) /boot/vmlinuz
- 4) What is the name of the pre-kernel gzip file located in /boot that helps the kernel load?
 - a) vmlinuz
 - b) initrd
 - c) initram
 - d) init
- 5) Where is the file location where the GNU Grub configuration is stored that a user would edit?
 - a) /boot/grub/grub.cfg
 - b) /etc/default/grub
 - c) /etc/grub/grub.cfg
 - d) /boot/kernel/conf
- 6) In the /etc/default/grub file, which of these options below would I edit to display the *splash* screen on boot so kernel messages are displayed?
 - a) GRUB_CMDLINE_LINUX_DEDFALT
 - b) GRUB_BACKGROUND
 - c) GRUB_GFXMODE
 - d) GRUB_TIMEOUT
- 7) What is the command to make changes to /etc/default/grub permanent?
 - a) No special command just edit and save /etc/default/grub
 - b) sudo apt-get update
 - c) sudo update-grub
 - d) sudo updatedb
- 8) Under SysVinit - what is the ancestor process that launches first and everyother process is started by it?
 - a) root
 - b) sbin
 - c) init
 - d) systemd
- 9) Under SysVinit - what runlevel is considered multi-user command-line only?
 - a) 1
 - b) m

- c) 3
 - d) 5
- 10) Under SysVinit - what runlevel is considered multi-user GUI only?
- a) 1
 - b) 0
 - c) 3
 - d) 5
- 11) Which company created the Upstart init system as an improvement of SysVinit?
- a) RedHat
 - b) Debian
 - c) Oracle
 - d) Ubuntu
- 12) What is the name of the init system that has replaced SysVinit in every single major Linux distribution (Not including Devuan and Gentoo Linux)?
- a) systemX
 - b) systemd
 - c) systemV
 - d) initrd
- 13) What is the name of the command you use in systemd to inspect, start, stop, and modify process states?
- a) systemd
 - b) systemd-init
 - c) service
 - d) systemctl
- 14) What would be the command to disable (make the service not start at boot time) the httpd service on Fedora using systemd?
- a) sudo service apache2 stop
 - b) sudo systemctl disable apache2.service
 - c) sudo systemctl stop apache2.service
 - d) sudo systemctl disable httpd.service
- 15) What is the Linux command to inspect processes (not part of systemd)?
- a) p
 - b) ps
 - c) proc
 - d) meminfo
- 16) SysVinit used the concept of PIDs and PPIDs—what did systemd replace these with?
- a) proc-groups
 - b) sys-groups
 - c) cgroups
 - d) xgroups
- 17) What is the signal name for a kill -2 command?
- a) SIGHUP
 - b) SIGINT
 - c) SIGKILL
 - d) SIGTERM
- 18) The /proc filesystem provides you what? (choose all that apply)
- a) Provides you a file based interface to the processes that are running on your system
 - b) It can be regarded as a control and information centre for the kernel
 - c) Shows up to the second process usage—updated in real time

- d) Is a replacement for the top command
- 19) What command can be used to list all the pci devices attached to your system?
- ls -pci
 - ls -p
 - lsusb
 - lspci
- 20) What is the runlevel target that has a single user only as root, using no password: commonly called single-user mode?
- runlevel3.target
 - runlevel5.target
 - runlevel0.target
 - runlevel1.target

17.9.2 Podcast Questions

View the presentation by FreeBSD developer Benno Rice from BSDCan 2018 at <https://www.youtube.com/watch?v=6AeWulfZ7bY> and answer the following questions:

- ~1:00 Who is Benno Rice?
- ~1:31 What is Contempt Culture?
- ~3:21 What is inits job?
- ~6:11 What lead to the concept of a service?
- ~8:35 What does the traditional rc system not do?
- ~9:27 What OS had a strong initial concept of services from the beginning?
- ~10:00 On MacOS what did launchd replace?
- ~11:53 In 2010 What was Lennart Poettering looking at?
- ~13:48 What other service did Lennart say he was heavily borrowing from?
- ~14:01 What does Lennart say that systemd is about?
- ~14:43 What is the layer in-between the kernel and the userspace created by systemd?
- ~17:11 Does systemd violate the UNIX philosophy?
- ~20:33 What does Benno think is incredible about what Lennart accomplished?
- ~25:26 Why is using systemd as a recruiting tool for BSD (which doesn't have it) a bad idea?
- ~28:20 What are a few features that BSD could gain from systemd?
- ~28:20 Why can't BSD run containers?

17.9.3 Lab

Objectives:

- Modify GRUB settings
- Use `systemctl` to start, stop, and examine processes in systemd
- Use `systemd-analyze` to understand what services are loading during system boot
- Change `systemd.targets`
- Use the `nice` command to modify a processes priority
- List kernel modules currently loaded on your Linux system

Outcomes:

At the conclusion of this lab you will be able to manage, edit, and list system processes in systemd—helping you to master the concepts of systemd.

Instructions: Make a folder in your Github repo named Week-13, create a file called ReadMe.md. In this file you will make a header H2 (## in markdown) for each question and will post a screenshot of the command and output that each question requires.

- Change the default grub settings in Ubuntu add a background image (preferably dark) and remove or disable the `quiet splash` option) make sure to execute `update-grub` before rebooting or changes won't be written.
- Use the `systemd-analyze` tools to print out the most recent boot time for your system

- 3) Install MariaDB server, `sudo dnf install mariadb`.
 - a) Use the command `systemctl status <servicename>` after MariaDB is installed to display its current status, then enable the service via `systemctl`, and then start the service. Now reboot your system.
- 4) With MariaDB enabled, use the `systemd-analyze` tools to print out the most recent boot time for your system again and compare if adding this service increased boot times.
- 5) Use `systemd-analyze blame` to collect start times of each element after installing and enabling the MariaDB service
- 6) Use `systemctl` to enable and start the `httpd.service` (Fedora)
- 7) Use `systemctl` to `SIGHUP` the `httpd.service` (Fedora)
- 8) Change the `systemd` target to the `systemd` commandline only level, display the `systemd` default target level - then change back to the GUI target (or `runlevel5`).
- 9) Using `systemctl` and the `--show` option, display the “After” and “Wants” of the `sshd.service`
- 10) `nice` a command - create/compile a C infinite loop program and `nice` it to lowest priority and then highest priority. Open a second terminal tab/window and use `htop` (install it if needed) to display that process’ system usage
- 11) Launch multiple tabs in Firefox using this command:
 - a) `firefox -new-tab -url krebsonsecurity.com -new-tab -url twit.tv/floss/`. Find the process IDs via `ps -ef` and kill those tabs/processes with a `kill -2` command
 - b) Repeat the above launch command and this time use `systemd` and the proper `cgroups` to kill the FireFox processes
- 12) Using `lsmod` and `grep` list all of the kernel modules loaded on your system that contain VirtualBox (search for `vb*`).
- 13) Run the `systemd-systemctl` command to list the VirtualBox kernel modules that are loaded
- 14) Run the command that will list all the PCI devices attached to your system
- 15) Type one of the two commands mentioned in the chapter to display info about your CPU hardware
- 16) Using `systemctl` find the `cgroup` for the `apache2` webserver (known as `httpd` on Fedora) and issue a `SIGHUP` to the entire `cgroup`.
- 17) Using `systemd-cgls` list and filter (`grep`) and show the sub-process IDs for the `httpd.service`
- 18) Use the `timedatectl` command to change the clock of your system to UTC.
- 19) Use the `hostnamectl` command to:
 - a) `set-hostname` to `itmo-556-xyz` (`xyz` is your initials)
 - b) `set-location` to: `d1r1u22`
 - c) `set-chassis` to: `vm`
 - d) `set-deployment` to: `development`
- 20) Install a copy of Devuan Linux devuan.org. Take a screenshot of the `ps -ef` command focusing on PID 1.
 - a) Install the `openRC` init system via `sudo apt-get install openrc`
 - b) The install process asks you to run a command after successful install: `for file in /etc/rc0.d/K*; do s=`basename $(readlink "$file")` ; /etc/init.d/$s stop; done` explain what this command is doing.
 - c) Reboot the system and take a screenshot of the output of the `ps -ef` command focusing on PID 1.
- 21) What would be the command to change the `systemd` target `runlevel` to `recovery` mode? Execute this command and take a screenshot of the result.
- 22) Review the content of the `mysql.service` file, list the contents of the `[Install]` header that must load before and after the `mysql` service starts.
- 23) Using `GCC` `sudo dnf install gcc` or `sudo apt-get install build essential`, create and compile a simple C++ code that is an infinite loop – just put `while true` in the body of `main`. Execute this script (note you could do this in Python as well, your choice). Use `systemd-cgtop` to display the usage and capture that output.
 - a) use the `ps` and various filters to show only information related to this process.
 - b) use the `systemd-cgls` command and filters to display the process information
 - c) use the `kill` command from `systemd` to kill the `cgroup` related to the infinite loop process.

17.10 Chapter 11

- 1) What is the `fdisk` program?

- a) a dialog-driven program for the creation and manipulation of partition tables.
 - b) a filesystem creation tool
 - c) a tool for formatting floppy disks
 - d) a tool for removing disks from a system
- 2) What is the default Virtual Box disk type?
- a) VDI
 - b) HDD
 - c) COW
 - d) VDD
- 3) After attaching a new virtual disk what is the next step?
- a) partitioning
 - b) make a filesystem
 - c) mount a filesystem
 - d) add an extent
- 4) Which command will print out currently all the block devices, their device name, and their partitions in a nice tree based format.
- a) lspci
 - b) lsblk
 - c) lsusb
 - d) lstree
- 5) fdisk is part of what package?
- a) utils
 - b) GNU
 - c) utils-linux
 - d) utils-unix
- 6) What would be the name of the second sata disk attached to your system?
- a) sda
 - b) sdb
 - c) sdc
 - d) sdd
- 7) What is the name of the first native Linux filesystem released in 1992?
- a) ext2
 - b) minix
 - c) ext4
 - d) ext
- 8) What is the name of the current default Linux Filesystem?
- a) ext
 - b) btrfs
 - c) ext3
 - d) ext4
- 9) Ext4 breaks up data into _____, which is the smallest sized piece of data that can be read or written?
- a) sectors
 - b) tracks
 - c) blocks
 - d) clusters
- 10) If you use the ext2 filesystem and choose a 4 KiB block, what is the maximum filesystem size?
- a) 2 TiB
 - b) 16 TiB

- c) 4 TiB
 - d) 4 GiB
- 11) What is the name of the maintainer of the ext4 filesystem?
- a) Brian Kernighan
 - b) Theodore Ts'o
 - c) Andrew Tanenbaum
 - d) Google
- 12) What is the name of the filesystem that the ext4 maintainer, Theodore Ts'o, is recommending to replace ext4?
- a) XFS
 - b) Btrfs
 - c) ZFS
 - d) HAMMER
- 13) What is the name of the filesystem that RedHat adopted on their RHEL 7 platform to replace ext4 and support better performance on large filesystems?
- a) ZFS
 - b) XFS
 - c) Btrfs
 - d) HAMMER
- 14) What is the name of the package needed to install on Ubuntu to be able to create XFS filesystems?
- a) `sudo apt-get install xfsprogs`
 - b) `sudo apt-get install zprogs`
 - c) `sudo apt-get install file-progs`
 - d) `sudo apt-get install zfsprogs`
- 15) What is the name of the combined filesystem and logical volume manager designed by Sun Microsystems?
- a) XFS
 - b) SunFS
 - c) ZFS
 - d) Btrfs
- 16) Which command is valid for making an ext4 filesystem on a partition `/dev/sdb1`
- a) `sudo mkfs.ext4 /dev/sdb`
 - b) `sudo mkfs.ext4 /dev/sdb1`
 - c) `sudo mkfs /dev/sdb1`
 - d) `sudo makefs`
- 17) What is the command to mount an ext4 filesystem, `/dev/sdb1` on a mount point `/mnt/data-drive-2`?
- a) `sudo mnt /dev/sdb1 /mnt/data-drive-2`
 - b) `sudo mnt -t ext4 /dev/sdb1 /mnt/data-drive-2`
 - c) `sudo mount -t ext4 /dev/sdb1 /mnt/data-drive-2`
 - d) `sudo mount /dev/sdb1 /mnt/data-drive-2`
- 18) What is the command used to list the current mountpoints, that will be mounted automatically at boot?
- a) `/etc/mstab`
 - b) `/etc/default/grub.conf`
 - c) `/etc/fstab`
 - d) `/etc/tab`
- 19) What is the command used to create a LVM physical volume?
- a) `pvcreate`
 - b) `pvck`
 - c) `pvadd`
 - d) `pvscan`

20) What is the command used to create a LVM volume group?

- a) vgcreate
- b) vgck
- c) vgdisplay
- d) vgmknodes

17.10.1 Podcast Questions

DragonFly BSD - Listen to this podcast: <https://ia802605.us.archive.org/9/items/bsdtalk248/bsdtalk248.mp3>

- ~1:25 What did DragonFly BSD drop with the 4.0 release?
- ~1:40 What was the other major feature that DragonFly BSD added?
- ~3:40 What modification did they add to the Packet Filter?
- ~10:00 What is the largest system DragonFly BSD has access to?
- ~11:45 What is the difference between DragonFly BSD's network stack compared to BSD and Linux?
- ~13:25 What are the limitations of the Hammer 1 Filesystem?
- ~13:45 What features will Hammer 2 Filesystem add?
- ~15:45 What is the intended use case of Hammer 2 FS?
- ~18:00 What sub-system is still in the works needed to make DragonFly BSD a stable work station?
- ~25:00 What is package-ng?
- ~30:00 How does DragonFly BSD handle suspend and resume functions common to laptops?
- ~35:50 What is the growing issue about systemd in relation to BSD?
- ~38:00 Of the 20,000 packages available in DragonFly BSD where are they primarily targeted?
- ~38:30 Out of FreeBSD, OpenBSD, NetBSD, and DragonFly – what is each project focusing on?
- ~40:23 How does GPL based Linux software cross over into BSD distros?

17.10.2 Lab

Objectives

- Creating virtual disks in Virtual Box
- Creating new partitions in fdisk
- Creating new filesystems with mkfs
- Creating new filesystems in ZFS
- Mounting new filesystems
- Editing `/etc/fstab` to make our mounts permanent

Outcomes

At the conclusion of this lab you will have successfully created a new virtual disk in Virtual Box, created new partitions using fdisk, formatted those partitions using mkfs, XFS, and ZFS, and mounted all those partitions manually and automatically using the `/etc/fstab`.

1. Create 1 virtual drive
 - a. Use fdisk to create a primary partition
 - b. Format it with ext4
 - c. Mount it to `/mnt/disk1`
 - d. Add it to your fstab
2. Create 2 virtual drives
 - a. Create a single volume group named `vg-group`
 - b. Create 1 logical volume named `lv-group`
 - c. Format it with XFS
 - d. Mount it to `/mnt/disk2`
 - e. Add the `lv-group` to your fstab
3. Using the same LVM as before
 - a. add an additional virtualbox disk and then create a LVM physical disk
 - b. Grow the volume group and logical volume

- c. Grow the XFS file system
4. Using LVM of the previous exercise on the logical volume lv-group create a 25 mb text file named datadump.txt
 - a. Following this tutorial: <http://tldp.org/HOWTO/LVM-HOWTO/snapshotintro.html> create an LVM snapshot of the logical volume named lv-backup
 - b. Mount the snapshot to /mnt/disk3 (create this location if not existing)
 - c. `ls -l` the contents of /mnt/disk3
5. Install a copy of FreeBSD 11
 - a. Attach two additional virtual disks
 - b. Create a zpool stripe containing both disks
 - c. Execute a `zpool status` command to display the contents of the zpool
6. Using Ubuntu 18.04 set networking to bridged mode (take note of your public IP by typing: `ip a sh`)
 - a. Attach a virtual disk
 - b. Using this tutorial: <https://www.hiroom2.com/2018/05/05/ubuntu-1804-tgt-en/> configure the system using as an iSCSI target
 - c. Use the proper `iscsi` command to list the current targets
7. Using a second Ubuntu 18.04 instance with its network mode set to bridged (note the public IP)
 - a. Using this tutorial: <https://help.ubuntu.com/lts/serverguide/iscsi-initiator.html> configure and complete iSCSI initiator
 - b. List the currently available iSCSI targets on your network
 - c. Create two files on the connected iSCSI target - file1.txt and file2.txt and list those files
8. Create 3 Virtual disks and install the ZFS package
 - a. Attach it to an existing Ubuntu 18.04 system
 - b. Create a zpool stripe with two disks name it datapool
 - c. Execute a `zpool status` and a `zpool list` command
 - d. Expand the capacity of the zpool by adding the third disk in
 - e. Execute the `zpool status` command
 - f. Now take the first disk out of the zpool
 - g. Execute the `zpool status` command
9. From the previous exercise using your ZFS pool named datapool create a 25 megabyte file named datadump.txt
 - a. Attach a third virtual disk to the system and create a zpool named backup
 - b. Execute the `ls -l` command to display the file and its size
 - c. Take a ZFS snapshot of the datapool named @today
 - d. Using the ZFS `send` and `recv` commands copy the @today snapshot to the zpool named backup
 - e. Execute `ls -l` command on the zpool backup
 - f. Using the commandline, append an additional 25 mb to /datapool/datadump.txt
 - g. Execute an `ls -l` on zpool datapool and backup to compare the two files
10. On the same Ubuntu 18.04 system create a systemd mount.unit file for both ZFS partitions created in the previous exercise.
 - a. List both contents here
 - b. Reboot the system and make sure it works
11. Using the 2 Ubuntu 18.04 systems you used in exercises 7 and 8 create a 25 megabyte file named databasedump.txt on the zpool datapool
 - a. On the first system (the system without zpool datapool) create a datapool name backuppool (you might need to attach a virtual disk to do this)
 - b. Take a snapshot of the zpool datapool and name it @now
 - c. Execute the remote `send` and `recv` command over `ssh` to migrate the snapshot to the pool backuppool
 - d. You may need to exchange SSH keys via `ssh-keygen` and `ssh-copy-id` first

12. On the zpool named datapool on Ubuntu
 - a. Execute a `zpool status` command
 - b. Enable LZ4 compression on the zpool datapool
 - c. Execute a `zpool list` command to display that compression is enabled
13. On the zpool named datapool execute a `zpool status` command
 - a. Execute a scrub of the zpool datapool
 - b. Create a cron job that executes a `zfs scrub` on the zpool datapool at 3 am every Sunday morning
14. Using the sample from the text on your Ubuntu 18.04 system add two additional virtual disk
 - a. Create two partitions on each of these devices
 - b. Then using the sample code add these two devices as a log and a cache to the zpool datapool
 - c. Execute a `zfs status` command for the zpool named datapool
15. On your Fedora system execute any of the commands listed to print out the disk serial numbers
16. Research:
 - a. Using Newegg.com find the current price per Gigabyte for the following along with listing the throughput of the drive and make a chart of the results.
 - b. Seagate Barracuda 4 TB
 - c. Western Digital Blue 1 TB
 - d. Western Digital Red 10 TB
 - e. Samsung 970 EVO M.2 500 GB
 - f. Corsair Force MP300 M.2 960 GB
 - g. Intel Optane M.2 32 GB - Need to explain what 3D XPoint technology is

17.11 Chapter 12

17.11.1 Review Questions 12

1. Using the ip2 suite of tools, which command would show your IP address?
 - a. `ifconfig`
 - b. `ipconfig`
 - c. `ip address show`
 - d. `show address`
1. Using the ip2 suite of tools, which command would show your routing table?
 - a. `ss`
 - b. `route`
 - c. `ip route show`
 - d. `ip -r`
1. What tool could you use to establish if a server is responding to requests?
 - a. `pong`
 - b. `ping`
 - c. `google`
 - d. `traceroute`
1. What is the purpose of a NETMASK?
 2. What is the purpose of DNS?
 3. What is a valid class C address?
 - a. 10.0.0.0
 - b. 172.24.0.0
 - c. 192.168.1.0
 - d. 221.0.0.0
1. If you had a network with a CIDR block of /23 how many IP addresses would you have control of?

- a. 23
- b. 254
- c. 512
- d. 256

1. If you had a CIDR block of /24 and a network address of 192.168.1.0, how many IP addresses would you have?

- a. 10
- b. 0
- c. 24
- d. 256

1. How does CIDR block addressing differ from Class based networking (A-D)?

2. What tool is used to release a dhcp address from the command line?

- a. `rhclient`
- b. `ipconfig /release`
- c. `dhclient -r`
- d. `xclient -r`

1. using the ip2 suite, What tool can be used to monitor and examine all current local ports and TCP/IP connections?

- a. `ss`
- b. `net`
- c. `wireshark`
- d. `netstat`

1. Where are your network card settings located on Ubuntu while using network manager?

2. Where are your network card settings located on CentOS/Fedora using network manager?

3. Where are your network card settings located on Ubuntu 18.04 using netplan?

4. What are the three major opensource webservers?

- a. Apache, Nginx, and openhttpd
- b. httpd, Nginx, openhttpd
- c. Apache, IIS, Nginx
- d. Apache, Lighttd, Nginx

1. What are two related and major opensource relational databases?

- a. SQL and MySQL
- b. MariaDB and MySQL
- c. MySQL and Oracle DB
- d. Nginx and MySQL

1. Name one major No-SQL database mentioned in this chapter?

2. What is the file location that the system uses as a *local DNS* for resolving IPs?

- a. `etc/systemd/hostd`
- b. `/etc/hosts`
- c. `/etc/allow`
- d. `/etc/etc/etc`

1. What is the name of the file that you would place in your home directory, that allows you not to have to type your login password for a MySQL database?

- a. `~/my.cnf`
- b. `/etc/mysql/settings.conf`
- c. `~/allow`
- d. `~/my.cnf`

1. Before systemd, NIC interface naming schemes depended on a driver based enumeration process: they switched to a predictable network interface names process that depends on what for the interface names?

- a. driver loading order

- b. interface names depend on physical location of hardware (bus enumeration)
- c. kernel version
- d. What ever Lennart Poettering feels like naming them

17.11.2 Podcast Questions 12

View or listen to this Podcast about Nginx: <http://twit.tv/show/floss-weekly/283>

1. ~2:02 What is Nginx?
2. ~3:22 What percentage of the world's websites are served with Nginx?
3. ~4:57 What was the challenge that lead to the creation of Nginx?
4. ~5:33 What is the main architectural difference between Nginx and Apache web servers?
5. ~8:32 What are some of the main use cases for Nginx?
6. ~11:00 When did Sarah get involved in Nginx?
7. ~12:56 Where did Nginx come from?
8. ~17:41 What is "caching" in relation to websites?
9. ~19:45 What is "proxying" in relation to websites?
10. ~29:36 What was the founder's motive to open source Nginx?
11. ~34:00 What is the difference in the open source Nginx and the commercial version? (Freemium?)
12. ~40:19 Are there Linux Distro packages for Nginx?
13. ~53:10 Can Apache and Nginx co-exist or is it a winner take all?

17.11.3 Lab 12

1. Using two virtual machines, while powered off, in the VirtualBox settings, enable a second network interface and set the type to host-only (details are in last chapter and the VirtualBox networking details are in chapter 03).
 - i. You will be modifying the IP address of both of these systems: 192.168.33.10 and 192.168.33.11, netmask is 255.255.255.0 and gateway should be 10.0.2.2 (this is due to VirtualBox).
 - ii. Configure these settings in Network Manager for the respective Virtual Machines.
 - iii. Capture a screen shot of each system using the `ping` tool to ping the other IP and its results.
 - iv. Modify the `/etc/hosts` file and add an entry for both system in both systems
 - v. Execute the `ping` command again this time using the hostname declared in the `/etc/hosts` file.
1. Repeat the above exercise but deactivate NetworkManager in `systemctl` and activate `systemd-networkd` (CentOS)
 - i. In addition, on Ubuntu modify the Netplan conf to use `networkd` and place your YAML configuration there.
1. Using `firewalld` open port 22 permanently to allow SSH connections to your Fedora or Centos system.
2. Using `firewalld` open port 80 permanently to allow SSH connections to your Fedora or Centos system.
3. Enable the firewall to start at boot and show its status after a successful boot.
4. Install and enable `firewalld` on Ubuntu, deactivate UFW if it is running.
5. Create a self-signed SSL certificate
6. Create a sample PHP webpage that displays `phpinfo()` at `https://localhost/index.php`. Name the file `info.php` and push it to your GitHub repo
 - i. Make sure you have installed all the pre-reqs (Apache2 and PHP)
 1. Enable the Apache Webserver and the proper firewall port to serve you `phpinfo` page over `https`.
 2. Going to [Wordpress.org](https://www.wordpress.org) and download the latest tar.gz file. Follow the 5 minute setup to configure a working Wordpress blog.
 3. Repeat the install process above, this time using two servers, with static IP addresses configured, placing the MySQL database on a separate IP address – configuring Wordpress properly and installing all needed pre-reqs.
 - i. Make sure to open the proper firewall ports and note that the first server will be the webserver and requires the `apache2`, `php`, `php-mysql`, and the `php-client` library only.
 - ii. The second database server requires the `Mysql-server` package. Make one to be Ubuntu one to be Fedora/CentOS.
1. Install the Ghost Blogging Platform on a single OS of your choice.

- i. Install nodejs, nginx, and mysql prerequisites
 - ii. Create a directory in your home directory called **ghost-files**. Execute the install tutorial in the next step in that directory.
 - iii. Follow the install instructions at Ghost.org to install the Ghost blogging platform.
1. Create a fresh Ubuntu Virtual Machine and create a shell script that will automate the installation of the following and their dependencies:
 - i. Install and Active Firewall, open ports 22, 80, 443
 - ii. Download and install latest Wordpress tarball (look at using `sed -i` to do an inplace substitution of the default values in the wordpress config file)
 - iii. Firewall to only allow single IP connection for SSH and port 80 and 443 (that IP being a second virtual machine with a static IP assigned)
 - iv. Modify `/etc/hosts` to include these entries and take a screenshot of the output of your firewall status as well as you accessing the Wordpress blog via the browser using the hostname configured in `/etc/hosts` on the remote machine.

17.12 Chapter 13

17.12.1 Review Questions 13

1. What is a common title given to IT workers who are responsible for the ongoing operations of applications and their environments?
 - a) saints
 - b) devs
 - c) devops
 - d) ops
2. What would describe Mitchell Hashimoto's design goals in created Vagrant?
 - a) Automation
 - b) Separation of Duties
 - c) Profit
 - d) Inspection
3. What is the name of the tool originally built as an abstraction layer on top of VirtualBox to deploy virtual machines?
 - a) Packer
 - b) VirtualBox
 - c) Terraform
 - d) Vagrant
4. What is the name of the tool originally built as a way to automate the installation of any operating system into an artifact?
 - a) Packer
 - b) VirtualBox
 - c) Terraform
 - d) Vagrant
5. What year approximately was Vagrant introduced?
 - a) 2019
 - b) 2001
 - c) 2010
 - d) 2015
6. Fill in the blank. Think of Vagrant as _____ between you and VirtualBox, Hyper-V, Docker, or even VMware desktop
7. What is the name of the file type Vagrant uses that contains an `vmdk` and `ovf`?

- a) *.vdi
 - b) *.vhd
 - c) *.box
 - d) *.zip
8. Name the file that contains the configuration file for each Vagrant box file.
9. What is the correct command to add the Vagrant Box `centos/7`?
- a) `vagrant add box centos/7`
 - b) `vagrant box add centos/7`
 - c) `vagrant init centos/7`
 - d) `vagrant add centos/7`
10. What is the command used to list all Vagrant Boxes being managed by Vagrant?
- a) `vagrant list box`
 - b) `vagrant boxes list`
 - c) `vagrant box list`
 - d) `vagrant list`
11. What is the correct command to initialize a Vagrant file for Vagrant Box named `centos/7` that has already been added to the system?
- a) `vagrant init`
 - b) `vagrant init centos/7`
 - c) `vagrant box add centos/7`
 - d) `vagrant init 7`
12. What is the Vagrant command to start or turn on a Vagrant Box?
13. What is the Vagrant command to restart a Vagrant Box?
14. What is the Vagrant command to shutdown or poweroff a Vagrant Box?
15. For Packer.io, what is the descriptive name of the json file used for building artifacts?
- a) image template
 - b) machine.json
 - c) build template
 - d) provisioner
16. What is the name of the stage that runs after the builder portion of a build template?
- a) imager
 - b) provisioner
 - c) vboxmanage
 - d) post-processor
17. What is the name of the stage that runs after building and provisioning of Packer artifacts is complete?
- a) imager
 - b) provisioner
 - c) vboxmanage
 - d) post-processor
18. If there is an error in any part of the Packer build command what will happen?
- a) nothing
 - b) an error will be logged but the process will continue
 - c) the command will terminate and any artifacts will be deleted
 - d) the user will be prompted
19. What is the generic name of the file that is provided to Packer to help it complete the manual question part of the install?
- a) secret file

- b) answer file
- c) question file
- d) pxe file

20. What are the respective names of the RedHat and Debian based answer files?

- a) jumpstart and preseed
- b) kickstart and jumpstart
- c) kickstart and preseed
- d) Chef and Puppet

17.12.2 Podcast Questions 13

See the presentation at: <https://www.youtube.com/watch?v=xXWaECk9XqM>: The Container Revolution: Reflections After the First Decade by Bryan Cantrill.

1. ~0:30 Where does/did Bryan work, who recently bought that company, and what do they do?
2. ~1:33 What is the birth date of containers?
3. ~3:25 What was the next iteration of containers?
4. ~3:49 What is the purpose of a Jail?
5. ~5:10 What did SUN call their full application environment they created in 2002?
6. ~6:13 What is every application running on?
7. ~8:43 What did Amazon announce in 2006?
8. ~9:00 In 2006 what technology was Joyent using to run its Public Cloud? In 2006 what technology was Amazon using to run its Public Cloud?
9. ~9:25 What became de facto for the cloud?
10. ~11:18 What happens to the RAM when you give it to an operating system?
11. ~14:40 What does Joyent's Manta service allow you to do with containers and objects?
12. ~18:58 What command hadn't been created in 1986?
13. ~21:45 When did the world figure out containers and what was this product?
14. ~22:57 Why did the container revolution start with Docker?
15. ~24:07 Containers allow developers to do what?
16. ~26:00 What is Triton and what does it do?
17. ~31:42 What are the two approaches to the container ecosystem, and what is the difference?
18. ~33:25 What is the "Hashi" ethos?
19. ~37:00 What was the mistake that happened with the pilot operator release valve at 3 Mile Island?
20. ~39:05 With container based systems in what terms must we think in?
21. ~40:00 Why is scheduling containers inside of Virtual Machines a bad idea?
22. ~What are Joyent's thoughts regarding Virtual Machines in the application stack?

17.12.3 Lab 13

Lab goes here

17.13 Chapter 14

17.13.1 Podcast 14

Docker - <https://twit.tv/shows/floss-weekly/episodes/330>

1. ~3:37 What is the overview of Docker and what does it do?
2. ~4:58 Linux Containers already existed, what does Docker do differently (who are they targeting)?
3. ~6:45 Is Docker language specific?
4. ~7:39 How is Docker similar to a static binary?
5. ~10:15-12:00 How do Docker containers shift the line between developers and operations people?
6. ~13:30 How does the analogy of Docker and cargo shipping containers relate?
7. ~22:15 How does source code get from Git to a Docker Container?
8. ~22:50 Is a container like a server or a binary?
9. ~25:20 How does Docker handle container orchestration?

10. ~28:00 What does Docker Swarm do?
11. ~28:33 What does Docker Compose do?
12. ~32:00 What features does Docker not contain out of the box?
13. ~33:30 What is the process to build container images that are identical too each other?
14. ~36:01 What can you do with Docker IDs to make a clean predictable starting point?
15. ~37:45 How does Docker prevent builds from differing over a large number of machines?
16. ~42:15 Can Docker run containers that are different than the host OS and why?
17. ~48:23 What language is Docker written in?
18. ~51:00 How is Docker developed, what is the project's example?

17.13.2 Lab 14

17.14 Chapter 15

17.14.1 Podcast

Kubernetes

<https://twit.tv/shows/floss-weekly/episodes/351?autostart=false>

1. ~8:37 What problem is Kubernetes solving and how does it solve it?
2. ~9:16 What is a "PaaS" and what does it do?
3. ~10:20 Was Kubernetes built from the ground up as an opensource project?
4. ~12:35 How does Kubernetes handle IP addressing for containers?
5. ~14:35 What is the unit of abstraction with Kubernetes?
6. ~15:10 What does the word Kubernetes mean?
7. ~16:25 How does Kubernetes and Docker relate/intertwine?
8. ~24:00 What is the big selling point of using Kubernetes mentioned?
9. ~27:10 Does Kubernetes take care of all features needed to run an applicaiton or are additional software pieces needed?
10. ~28:55 What is the difference between an OS Container and a Virtual Machine?
11. ~31:01 How does Kubernetes deal with quality of service in relation to applications?
12. ~33:31 How does Kubernetes differ from SUN Solaris containers (zones)?
13. ~36:45 Does Kubernetes run on other cloud providers (Azure or AWS)?
14. ~37:42 How are customers using Kubernetes in production?
15. ~44:23 In what service does Kubernetes keep its system state in?
16. ~45:43 Why did Google decide to use its own Go language to develop Kubernetes?
17. ~52:02 What is the relationship between Google/Kubernetes and RedHat's OpenShift Paas?

17.14.2 Lab

Chapter 18

Appendix D - Git Tutorial

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE.	4 HOURS AGO
○	AAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Figure 18.1: *Git commit messages—after developing this book I understand this completely*

This is a tutorial for installing Git and configuring GitHub on Windows 10/11 and macOS.

18.1 Installing Git

Git is a piece of software that allows for distributed version control. [Version control](#) is an idea that starts with having a central repository for controlling access to source code. Version control is a way that all source code, media, documentation, and supporting scripts for a project can be stored in a central place—with its history and changes all managed and recorded. This is usually a remote location and referred to as the **single source of truth**. With the rise of the internet, the concept of DVCS—**distributed version control software**, implemented the bring-over, merge, and modify model. Currently the industry standard DVCS software is Git. Git should not be confused with GitHub. Git is opensource version control software and GitHub is a commercial implementation with a management portal for Git software, [owned by Microsoft](#).

Git can be installed on any operating system via an installer, but I recommend to install it via a third party package manger.

18.1.1 Windows 10 and 11 - Git Installation via Chocolatey

The [Chocolatey Windows 10 package manager](#) allows for scripted installs of applications. This tool is convenient for installing common applications such as Firefox, Android Studio, Java JDK, VS code, VirtualBox and other commonly installed tools. You need to enable PowerShell scripts, which is shown via [the install instructions](#). Using a package manager allows for having scripted installations as well as a function to update software in place from the command line.

From PowerShell (not CMD!) with Administrative privileges, run this command to install chocolatey:

```
Set-ExecutionPolicy Bypass -Scope Process -Force; `
[System.Net.ServicePointManager]::SecurityProtocol `
```



```
= [System.Net.ServicePointManager]::SecurityProtocol `
-bor 3072; iex ((New-Object `
System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))
```

Upon finishing this command you will need to close PowerShell and reopen it with Administrative privileges again. Once you have this done you can use the `choco` command to install Git amongst other software.

```
# from an admin console
choco install git vscode powershell-core
# You can also install VirtualBox via choco too
```

Once this is successful make sure to close the PowerShell console that was launched with Administrator privileges. Open PowerShell console, this time version 7—with the dark blue icon, not the light blue icon. Type the command: `git --version`, if the installation went well you will see version information printed to PowerShell similar to this: `git version 2.20.1.windows.1`

18.1.2 macOS - Git Installation via Homebrew

[Homebrew](#) is a third party package manager available for MacOS. Both Intel Macs and M1 Macs are supported by Homebrew. This functions as a much needed package manager and a way to install needed packages in an automated fashion. Using a package manager allows for having scripted installations as well as a function to update software in place from the command line.

To install Homebrew or `brew` run the below command:

```
/bin/bash -c `
"$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"
```

For installing Git on a Mac, open the `terminal` app. Run the command:

```
brew install git
brew cask install visual-studio-code
```

18.2 Git Basics

There are many Git Tutorials available on the Internet, some paid, and some free. The main source of documentation is the [Git Book](#) from the Git developers and is available on line for free and in a printed edition. We will be referencing some of that material in this tutorial and adding some more specific details.

18.2.1 Initial Git Setup

Once you have successfully installed the `git` program and an editor ([VS Code](#) is recommended) with first party Version Control integration you are ready to begin. As a note, though a good piece of software, we will not be using the GitHub Desktop software for this tutorial—everything can be done via your Shell and your editor.

You will have previously submitted your GitHub ID and received an invite to an private GitHub repo provided by the professor as part of their GitHub Organization. It will be in the form of: `https://github.com/illinoistech-itm/HAWKID`: accept this invitation. You will be greeted with a webpage that will show your **repository**, also known as a **repo**. This site, located on GitHub.com, will be your **source of truth** for all your markdown documents and code. From here we need to make a local copy of the repo so we can begin to interact with your remote repo.

18.2.1.1 Structure of a Git Repo

At first glance, Git and GitHub seem to just be an online file storage system, like Google Docs. The illusion of file storage is just for you to be able to visualize the repo contents in a familiar fashion. In reality [Git is not storing individual files](#), it is storing an original copy and then additional deltas or changes to that file, that can be applied to reconstruct your document. If you make 1000 changes to some source code file, your Git Repo is only storing the changes from the previous state, not 1000 copies of the file. This allows you to inspect the history of all the code committed to your repo as well as roll back in time to previous states in the history. Online file storage cannot do that.

18.2.1.2 Initial Setup

On your local system we now need to configure a few things. First we need to tell `git` a name and contact email so that every time code is committed to a repo, it is tagged with a name and a contact email. This is not so important when it is just you working on some homework, but is important when you are working on a large team or in a large company and want to know who made what change.

Go ahead and open a shell (PowerShell or Terminal on a Mac) and lets execute [a few setup commands for git](#).

```
# The first command is to set the user.name for each commit
git config --global user.name "John Doe"
# The second command is to set the user.email for each commit
git config --global user.email johndoe@example.com
```

This step only has to be done once on your computer after installation of `git`. You can check if the values are set correctly by issuing the command: `git config --list`

18.2.1.3 Git Commands

Git has a rich suite of tools and a large number of options. In this tutorial we are going to cover about seven of the core actions that will make you conversant with git as well as show how to securely setup remote access to your Git repo. You can always find help for commands by typing: `git --help` or for a specific command: `git add --help`. Note if you have a GUI a local webpage with the help info will open, if you are on a GUI-less server, then the text will print out to the screen.

- `git add`
 - Add file contents to the index
- `git commit`
 - Record changes to the repository
- `git status`
 - Show the working tree status
- `git push`
 - Update remote refs (remote repos) along with associated objects
- `git pull`
 - Fetch from and integrate with another repository or a local branch
- `git clone`
 - Clone a repository into a new directory
- `git remote`
 - Manage set of tracked repositories

18.2.1.4 Securing your GitHub authentication

In order to authenticate to your own repo we need to select an authentication method. In the past, GitHub used a username and password combo. This was not considered secure as the password and username had to be passed inside the URL of the request. We are all familiar with username and passwords and they are the easiest but the hardest to secure.

In October of 2021, GitHub, realizing how important it is to protect access to code repositories. Most large companies are using GitHub and if a hacker were able to get into your source code, this would be a valuable prize.

So GitHub removed the password authentication method, no longer possible. They replaced it with something called a PAT, [Personal Access Token](#). PATs have advantages over username/password combos.

- PATs can be declared, at creation time, as valid for a limited period
 - Default duration is 30 days
 - This cause keys to be cycled
 - Prevents old keys from having long lives – say after an employee leaves
 - Downside is that many keys have to be managed and distributed
- PATs can also limit the activity a key can do
 - Can be made read-only for instance
 - Can give you fine grain control over who can modify code in a repo

- PATs are actually one large mathematically unique tokens
 - No username or password that can identify you and your organization
 - Just a token
 - Downside - transmitted in your request
- Tokens are good for automated processes and are far better than usernames and passwords

We though are going to go one layer of security deeper. We will be using [Public-key cryptography](#) key pairs for SSH. We will be using the [Twisted Edwards Curve Cryptography](#) for the encryption.

18.2.1.5 Public-key cryptography key pairs

Some of these terms may be new to you, some may be familiar. In all cases we will walk you through each step regardless of your experience. The idea behind Public-key cryptography involves two keys to use for authentication in place of passwords and usernames. There is a public key, which is distributed widely and there is a private key, which completes the key and needs to be kept private.

The generation of such key pairs depends on cryptographic algorithms which are based on mathematical problems termed one-way functions. Effective security requires keeping the private key private; the public key can be openly distributed without compromising security.

Now lets make a key pair.

18.2.1.6 Generate a Public-key crypto key pair

We will be using the `ssh-keygen` command to generate your key pair. The command is the same on all platforms and will generate two keys for us, a **public key** and a **private key**. From PowerShell or the Mac or Linux Terminal run the following command:

```
ssh-keygen -t ed25519
```

A few things to watch out for, remember that Windows is **NOT** case sensitive, but Mac and Linux are case sensitive. Meaning of Mac, Git and git are not the same values, but on Windows they are.

```
# Note on Windows the divider: "\" is different than Mac and Linux
# Mac and Linux use the "/"
# This example is on a Windows system
# The username on Windows in this case is: palad
Generating public/private ed25519 key pair.
Enter file in which to save the key (C:\Users\palad\.ssh/id_ed25519):
# Mac example would look like this
# The username on the Mac in this case is: palad
Generating public/private ed25519 key pair.
Enter file in which to save the key (/Users/palad/.ssh/id_ed25519):
```

The value in the parenthesis is the default value. If you were to hit the “ENTER” key it would place your Public and Private Key files in the directory and name the file accordingly. Lets modify this entry so you can identify each key pair and its purpose later.

At the prompt we will type the following and it will look like this:

```
# Remember, No spaces in the file name!
# Windows
Enter file in which to save the key ...: C:\Users\palad\.ssh/id_ed25519_356_github_key
# Mac/Linux
Enter file in which to save the key ...: /Users/palad/.ssh/id_ed25519_356_github_key
```

The next prompt will ask you to enter a **passphrase**, this is an additional password that can be attached to a private key and would be required to use the private key for authentication. This can be a good idea as it is an extra layer of security against physical theft of your private key file. But in the use case we are working we are going to opt not to use it and handle security in a different method. In this case you can hit the “ENTER” key twice and it will not add a passphrase to your private key.

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

You will now see some output similar to this:

Your identification has been saved in c:\Users\palad/.ssh/id_ed25519_356_github_key.

Your public key has been saved in c:\Users\palad/.ssh/id_ed25519_356_github_key.pub.

The key fingerprint is:

SHA256:3LR4sEpKQgbA6LT7yOP54QcAUJ5/BaDEY/zgo3YWrOA palad@lenovo-laptop

The key's randomart image is:

```
+--[ED25519 256]--+
|0+o ...          |
|o**o   .         |
|+o0+    o .      |
|. *o+.  o * .    |
|o.=.+ o S +     |
|.E * + . .      |
|o =.o .         |
| +o...          |
|.ooo.           |
+-----[SHA256]-----+
```

We are interested in the first two lines of the output. They show the file location of the public and private key. The key fingerprint is not the content of the key. By default all operating systems, Windows, Mac, and Linux store key pairs in a hidden directory in your home directory called: `.ssh`.

Location of private and public keys -- note the public key has a .pub extension

The private key has no default extension

Your identification has been saved in c:\Users\palad/.ssh/id_ed25519_356_github_key.

Your public key has been saved in c:\Users\palad/.ssh/id_ed25519_356_github_key.pub

In order to make secure connections to GitHub we are going to need to add the content of the `.pub` Public key into your GitHub account. We can do this via copy and paste:

display the content of the .pub file -- note the location of the file

On Windows:

```
type c:\Users\palad/.ssh/id_ed25519_356_github_key.pub
```

On Mac/Linux

```
cat /Users/palad/.ssh/id_ed25519_356_github_key.pub
```

You will receive output similar to this

Don't worry the public key is meant to be openly distributed

```
ssh-ed25519 AAAAC3NzaC11ZDI1NTE5AAAAILOgJFa4p2bLzbiqSfin87zzrFC29vULvMXd+MrwHbL0
```

```
palad@lenovo-laptop
```

18.2.1.7 Add Your Public Key to GitHub

Now that we have a public key generated, copy the displayed content. We will now access the private GitHub repo assigned to you. Mine would be: <https://github.com/illinoistech-itm/jhajek>. This site is marked public so anyone can view it. Your repo will be a private repo that only you and the instructor can see and requires authentication to access.

The steps listed here will help add your Public Key to your GitHub account for authentication. To begin, select the circle icon in the upper right corner to expand the option menu.

In this new menu near the bottom of the list your will see the **Settings** option.

A new set of settings options will appear, we want to select on the left-hand side middle of the page, **SSH and GPG Keys**.

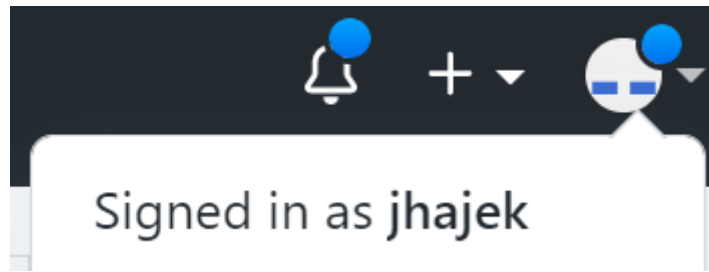


Figure 18.2: *Log into your GitHub Account*



Figure 18.3: *Select the Settings Option*

On the right hand side of the new page, there will be a green **New SSH Key** button.

You will be presented with two text-boxes. The first is a title – this is a comment that you can enter so you know where this key came from. I would recommend titling it based on two things: where is the key located (your laptop or desktop) and what is it for (the class number). This way you will give yourself a clue to the private keys location if you ever forget.

Paste the contents of the `.pub` file into the box labeled **Key** and hit the submit button. Your public key is now associated with your GitHub account to be used for authentication. Submit the key by hitting the green **Add SSH key** button.

SSH and GPG keys

Figure 18.4: *Select the SSH and GPG Keys option*



Figure 18.5: *Select Green New Key Button*

Title



Figure 18.6: *Enter a descriptive title for the key*

18.2.1.8 Create the config file in your .ssh directory

The last thing we need to do is create a file named `config` inside of your `.ssh` directory. This file is a shortcut convention where you can store values related to making SSH connections. This saves time typing on the command line all of the values needed to use our Private Key when we authenticate to GitHub. We will initiate this key pair via SSH. For those not familiar with SSH:

The Secure Shell Protocol (SSH) is a cryptographic network protocol for operating network services securely over an unsecured network.[1] Its most notable applications are remote login and command-line execution.

The `.ssh` directory is located in your home directory. For Windows this will be the path, `C:\Users\YOURUSERNAME\.ssh` on Mac and Linux it will be `/Users/YOURUSERNAME/.ssh`. On both operating systems you can use the `~` sign as a shortcut.

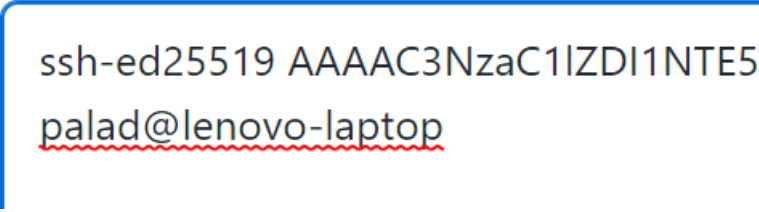
Let us create our `config` file. From the shell, lets type the command to open VS Code and create a file named, `config` in our `.ssh` directory.

```
# This command changes your directory to the correct location
cd ~/.ssh
# This command uses VS Code to create a file named: config
code config
```

Once VS Code is open you can now edit the `config` file. We need to put four entries into this file, though there are more values and entries that can be added, for now these four will do. The file is case-sensitive:

```
# Comments are allowed by using the # sign
# This line tells SSH to use these values whenever a connection is made
# to github.com
Host github.com
  # This is the username or ID you created for GitHub
```

Key



```
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5
palad@lenovo-laptop
```

Figure 18.7: *Paste the content of your Public key*

```
User jhajak
Hostname github.com
# This command tells SSH which Private key to use when making an SSH
# connection to GitHub
IdentityFile ~/.ssh/id_ed25519_356_github_key
```

18.2.1.9 Cloning via SSH

Now we are set and ready to clone your remote repository to your local repository. To do this we need to open the shell on your computer. One of the hardest things in computing is *naming things*. This includes where to store them. For instance should you use the Documents directory and create a new folder based on the class name? Some people prefer to create a folder per-semester. Whichever way you choose to do it remember a few things:

1. Be consistent in your naming and in the location your are storing things
2. Try not to use the Desktop folder – have a dedicated storage location so you can reason about where you put things
3. Try not to use iCloud or OneDrive based disks that are backed up to the cloud as you already have a remote repository
4. No spaces in filenames or folder names...[ever](#)

In my case I will create a new directory named `itmo-356` under my documents directory. You can use the File Explorer or the Finder in Mac to create files, but it is preferable to get comfortable with the commandline or shell. Opening PowerShell (or Terminal on a Mac) you will enter a series of commands:

```
# The cd command changes directory -- navigates down the tree
cd Documents
# The mkdir command makes a new directory
mkdir itmo-356
# The cd command moves you into the directory you just created
cd itmo-356
# This is the command that we use to make a clone of our remote repository
# Note: your URL will be different, change accordingly
git clone git@github.com:illinoistech-itm/sample-student.git
# Issue the cd command into the repo you just cloned (the name before the .git)
cd sample-student
# Issue the ls command to list the content of the just cloned directory
ls
# What do you see?
```

18.2.1.10 Opening your Repository to work with

Git is a commandline tool and it's many commands and features are replicated within the development IDEs such as VS Code, Atom, or Sublime editors. In our case we will be using VS Code. Our first step is to open the repository you just cloned to your system.

Once VS Code is open, click the **File** option and select **Open Folder**. Git doesn't see files, that is a concession to humans, who understand files, Git only sees **repositories** so to work with Git we always need to open a folder. In this case I will navigate to `~/Documents/itmo-356/sample-student` and click the **Select Folder** button. Remember, we are not opening individual files—we are opening a repository.

VS Code will open your just cloned repository, you will see an empty `Readme.md` file in the file manager (upper left) lets select this and begin to edit this file. The tutorial assignment requires you to edit your `Readme.md` to make a markdown document. This `Readme.md` will contain these elements written in Markdown. This is the [Markdown cheat sheet reference](#).

The document will include:

- h1 with your name
 - a picture of you
 - a picture of something that describes you
- h2 Where are you from?

- Tell us where you are from
- h2 IT Interests
 - Tell us what you IT Interests and or skills are
- h2 Something Interesting About You
 - Tell us something interesting about you
- What was your first computing device?

Here is a completed sample: <https://github.com/illinoistech-itm/jhajek/blob/master/README.md>. **Note**, I will have more folders than you because I have sample code for more classes.

18.2.1.11 Git commands

Git has a concept of code repositories. Changes to code are stored only as the changes plus the originals. Each time code is committed, you are not storing a new version of a file, but only the deltas from each change. Git can reconstruct the original file by applying all the changes to present you with the current file. After a `git clone` command you now have a local copy of your remote repo. The repository located on GitHub is called your **remote repo**.

Most Git commands execute on your local repo. As files are changed and you save these changes, you will need to first commit your code to your local repo. Each time a change of code is committed you must add a commit message. This is a 140 character note that you leave to yourself or team members to briefly explain what changes were made. Once the commit message has been added and the changes have been committed. The last step is to do a `git push` command which will push your deltas (changes) to the **remote repo** and synchronize the local and the remote repos.

The basic commands we will be using in this tutorial are as follows:

- `git clone`
- `git add`
- `git commit`
- `git push`

18.2.1.12 Committing code to your remote repo

Once your Markdown document, `Readme.md` has been created and updated you will notice in VS Code that a number in a blue circle appears over the third icon down. This is the source control management pane. Let's click on this icon. You will see under the **Source Control** header a message box, a commit button, and a Changes list. Go ahead and add a short **commit message** and hit the **Commit** button (or `Ctrl + Enter` as a shortcut). This will commit your changes to the local repository. The final step is to click the three triple dots next to the refresh icon above the Message box. Select the **Push** option, this will **push** your changes to the remote GitHub repo.

You can see your changes synchronized to the remote repository by visiting the URL to your private GitHub repository. Go ahead and complete creating the Markdown tutorial.

18.2.1.13 Create a `.gitignore` file

Every Git repository needs a `.gitignore` file. This file tells Git to ignore certain files. These are files that are too large, binary format, or things like security keys and passwords, that we don't want to be committing to our Git Repos. We will create a file named: `.gitignore` and place the following values into it and add, commit, and push the file to your private repo.

Files, Folders, security keys, and Binaries to ignore

```
*.vdi
*.box
.vagrant/
*console.log
packer-cache/
packer_cache/
*.pem
*~$
*.ova
```



```
output*/
vagrant.d/
*.iso
variables.pkr.hcl
*.priv
variables.json
.DS_Store
id_ed25519*
id_rsa*
id_rsa.pub
.Vagrantfile*
Vagrantfile~
config.default.yml
```

18.2.1.14 Conclusion

In this tutorial we introduced the concepts of Version Control and Public Key Encryption. We introduced using Git and GitHub to facilitate our use of version control. Then we introduced how to add Public Key Encryption for accessing our remote repos via SSH. We learned to configure our local systems and local repos and we cloned a private repo using Git. Finally we learned and created a markdown document and a .gitignore file, committed changes to the local repo, and successfully pushed our changes to the remote repository.