

LFS301

Linux for System Administrators

Version 7.2



Version 7.2

© Copyright the Linux Foundation 2021. All rights reserved.

For the exclusive use of LFS301 corp class taught 06 to 09 December

© Copyright the Linux Foundation 2021. All rights reserved.

The training materials provided or developed by The Linux Foundation in connection with the training services are protected by copyright and other intellectual property rights.

Open source code incorporated herein may have other copyright holders and is used pursuant to the applicable open source license.

The training materials are provided for individual use by participants in the form in which they are provided. They may not be copied, modified, distributed to non-participants or used to provide training to others without the prior written consent of The Linux Foundation.

No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without express prior written consent.

Published by:

the Linux Foundation

<https://www.linuxfoundation.org>

No representations or warranties are made with respect to the contents or use of this material, and any express or implied warranties of merchantability or fitness for any particular purpose or specifically disclaimed.

Although third-party application software packages may be referenced herein, this is for demonstration purposes only and shall not constitute an endorsement of any of these software applications.

Linux is a registered trademark of Linus Torvalds. Other trademarks within this course material are the property of their respective owners.

If there are any questions about proper and fair use of the material herein, please contact info@linuxfoundation.org.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Contents

1	Introduction	1
1.1	Linux Foundation	2
1.2	Linux Foundation Training	4
1.3	Linux Foundation Certifications	7
1.4	Linux Foundation Digital Badges	10
1.5	Laboratory Exercises, Solutions and Resources	11
1.6	E-Learning Course: LFS201	13
1.7	Distribution Details	14
1.8	Labs	21
2	Linux Filesystem Tree Layout	23
2.1	One Big Filesystem	24
2.2	Data Distinctions	25
2.3	FHS Linux Standard Directory Tree	26
2.4	root (/) directory	29
2.5	/bin	30
2.6	/boot	32
2.7	/dev	33
2.8	/etc	34
2.9	/home	35
2.10	/lib and /lib64	37
2.11	/media	38
2.12	/mnt	39
2.13	/opt	40
2.14	/proc	41
2.15	/sys	43
2.16	/root	44
2.17	/sbin	45
2.18	/srv	47
2.19	/tmp	48
2.20	/usr	49
2.21	/var	50
2.22	/run	51
2.23	Labs	52
3	Processes	55
3.1	Programs and Processes	56
3.2	Process Limits	60
3.3	Creating Processes	63
3.4	Process States	65
3.5	Execution Modes	66
3.6	Daemons	69
3.7	niceness	70
3.8	Libraries	72
3.9	Labs	75

For the exclusive use of LFS301 corp class taught 06 to 09 December

4 Signals	79
4.1 Signals	80
4.2 Types of Signals	81
4.3 kill	82
4.4 killall and pkill	83
4.5 Labs	85
5 Package Management Systems	89
5.1 Why Use Packages?	90
5.2 Software Packaging Concepts	91
5.3 Package Types	92
5.4 Available Package Management Systems	93
5.5 Packaging Tool Levels and Varieties	94
5.6 Package Sources	95
5.7 Creating Software Packages	96
5.8 Revision Control Systems	97
5.9 Available Source Control Systems	98
5.10 The Linux Kernel and git	99
5.11 Labs	101
6 RPM	103
6.1 RPM (Red Hat Package Manager)	104
6.2 Package File Names	105
6.3 RPM Database and Helper Programs	106
6.4 Queries	107
6.5 Verifying Packages	108
6.6 Installing and Removing Packages	109
6.7 Updating, Upgrading and Freshening RPM Packages	111
6.8 Upgrading the Linux Kernel	113
6.9 rpm2archive and rpm2cpio	114
6.10 Labs	115
7 dpkg	117
7.1 DPKG (Debian Package)	118
7.2 Package File Names and Source	119
7.3 DPKG Queries	120
7.4 Installing/Upgrading/Uninstalling	121
7.5 Labs	122
8 dnf and yum	123
8.1 Package Installers	124
8.2 dnf and yum	125
8.3 yum	126
8.4 Queries	127
8.5 Installing/Removing/Upgrading Packages	128
8.6 Additional dnf Commands	129
8.7 Labs	130
9 zypper	133
9.1 zypper	134
9.2 Queries	135
9.3 Installing/Removing/Upgrading Packages	136
9.4 Additional zypper Commands	137
9.5 Labs	138
10 APT	141
10.1 APT	142
10.2 APT Utilities	143
10.3 Queries	144

For the exclusive use of LFS301 corp class taught 06 to 09 December

10.4	Installing/Removing/Upgrading Packages	145
10.5	Cleaning Up	146
10.6	Labs	147
11	System Monitoring	149
11.1	System and Network Monitoring	150
11.2	sar **	152
11.3	System Log Files	154
11.4	Labs	156
12	Process Monitoring	159
12.1	Process Monitoring	160
12.2	ps	161
12.3	pstree	163
12.4	top	164
12.5	Labs	166
13	Memory Monitoring and Usage	169
13.1	Memory Monitoring and Tuning	170
13.2	/proc/sys/vm	172
13.3	vmstat	173
13.4	Out of Memory Killer (OOM)	174
13.5	Labs	176
14	I/O Monitoring and Tuning	177
14.1	I/O Monitoring	178
14.2	iostat	179
14.3	iotop	180
14.4	ionice **	181
14.5	Labs	182
15	I/O Scheduling **	185
15.1	I/O Scheduling	186
15.2	I/O Scheduler Choices	187
15.3	Labs	188
16	Linux Filesystems and the VFS	191
16.1	Filesystem Basics	192
16.2	Filesystem Concepts	193
16.3	Virtual Filesystem (VFS)	195
16.4	Available Filesystems	196
16.5	Journalling Filesystems	198
16.6	Special Filesystems	199
16.7	Labs	200
17	Disk Partitioning	203
17.1	Common Disk Types	204
17.2	Disk Geometry	205
17.3	Partitioning	206
17.4	Partition Tables	208
17.5	Naming Disk Devices	210
17.6	blkid and lsblk	211
17.7	Sizing up partitions	213
17.8	Backing Up and Restoring Partition Tables	214
17.9	Partition table editors	215
17.10	fdisk	216
17.11	Labs	217
18	Filesystem Features: Attributes, Creating, Checking, Mounting	223

For the exclusive use of LFS301 corp class taught 06 to 09 December

18.1	Extended Attributes	224
18.2	Creating and formatting filesystems	225
18.3	Checking and Repairing Filesystems	226
18.4	Mounting filesystems	227
18.5	NFS	231
18.6	Mounting at Boot and /etc/fstab	232
18.7	automount	234
18.8	Labs	236
19	Filesystem Features: Swap, Quotas, Usage	241
19.1	Filesystem Usage	242
19.2	Disk Usage	243
19.3	Swap	244
19.4	Filesystem Quotas **	245
19.5	Labs	251
20	The Ext4 Filesystems	255
20.1	ext4 Features	256
20.2	ext4 Layout and Superblock and Block Groups	257
20.3	dumpe2fs	260
20.4	tune2fs	261
20.5	Labs	262
21	The XFS and BTRFS Filesystems **	265
21.1	XFS	266
21.2	btrfs	268
21.3	Labs	269
22	Encrypting Disks	271
22.1	Filesystem Encryption	272
22.2	LUKS	274
22.3	cryptsetup	275
22.4	Using an Encrypted Partition	276
22.5	Mounting at Boot	277
22.6	Labs	278
23	Logical Volume Management (LVM)	281
23.1	Logical Volume Management (LVM)	282
23.2	Volumes and Volume Groups	283
23.3	Working with Logical Volumes	284
23.4	Resizing Logical Volumes	287
23.5	LVM Snapshots **	288
23.6	Labs	289
24	RAID **	291
24.1	RAID	292
24.2	RAID Levels	293
24.3	Software RAID Configuration	294
24.4	Monitoring RAIDs	295
24.5	RAID Hot Spares	296
24.6	Labs	297
25	Kernel Services and Configuration	299
25.1	Kernel Overview	300
25.2	Kernel Configuration	301
25.3	Kernel Boot Parameters	302
25.4	sysctl	303
25.5	Labs	304

For the exclusive use of LFS301 corp class taught 06 to 09 December

26 Kernel Modules	307
26.1 Kernel Modules	308
26.2 Module Utilities	310
26.3 modinfo	312
26.4 Module Configuration	313
26.5 Labs	314
27 Devices and udev	315
27.1 udev and Device Management	316
27.2 Device Nodes	317
27.3 Rules	320
27.4 Labs	323
28 Virtualization Overview	325
28.1 Introduction to Virtualization	326
28.2 Hosts and Guests	328
28.3 Emulation	329
28.4 Hypervisors	331
28.5 libvirt	335
28.6 QEMU	337
28.7 KVM	340
28.8 Labs	342
29 Containers Overview	353
29.1 Containers	354
29.2 Application Virtualization	355
29.3 Containers vs Virtual Machines	356
29.4 Docker	357
29.5 Docker Commands	359
29.6 Podman	360
29.7 Labs	361
30 User Account Management	365
30.1 User Accounts	366
30.2 Management of User Accounts	368
30.3 Locked Accounts	370
30.4 Passwords	371
30.5 /etc/shadow 	372
30.6 Password Management	374
30.7 Password Aging	375
30.8 Restricted Shells and Accounts **	376
30.9 The root Account	378
30.10 SSH	379
30.11 Labs	382
31 Group Management	385
31.1 Groups	386
31.2 Group Management	387
31.3 User Private Groups	388
31.4 Group Membership	389
31.5 Labs	390
32 File Permissions and Ownership	393
32.1 File Permissions and Ownership	394
32.2 File Access Rights	395
32.3 chmod, chown and chgrp	396
32.4 umask	399
32.5 Filesystem ACLs	400
32.6 Labs	401

For the exclusive use of LFS301 corp class taught 06 to 09 December

33 Pluggable Authentication Modules (PAM)	405
33.1 PAM (Pluggable Authentication Modules)	406
33.2 Authentication Process	407
33.3 Configuring PAM	408
33.4 LDAP Authentication **	409
34 Network Addresses	411
34.1 IP Addresses	412
34.2 IPv4 Address Types	413
34.3 IPv6 Address Types	415
34.4 IP Address Classes	416
34.5 Netmasks	417
34.6 Hostnames	418
34.7 Labs	419
35 Network Devices and Configuration	421
35.1 Network Devices	422
35.2 ip	423
35.3 ifconfig	425
35.4 Predictable Network Interface Device Names	427
35.5 Network Configuration Files	428
35.6 Network Manager	429
35.7 Routing	434
35.8 DNS and Name Resolution	437
35.9 Network Diagnostics	440
35.10 Labs	444
36 Firewalls	449
36.1 Firewalls	450
36.2 Interfaces	453
36.3 firewalld	455
36.4 Zones	457
36.5 Source Management	461
36.6 Service and Port Management	462
36.7 Labs	464
37 System Startup and Shutdown	467
37.1 Understanding the Boot Sequence	468
37.2 Boot Loaders	470
37.3 System Configuration Files in /etc	471
37.4 Shutting Down and Rebooting	474
37.5 Labs	475
38 GRUB	477
38.1 The Grand Unified Boot Loader (GRUB)	478
38.2 Interactive Selections with GRUB at Boot	480
38.3 Installing GRUB	481
38.4 Customizing the GRUB Configuration	483
38.5 Boot Loader Specification Configuration (BLSCFG)	484
38.6 Labs	486
39 System Init: systemd, SystemV and Upstart	487
39.1 The init Process	488
39.2 Startup Alternatives	489
39.3 systemd	490
39.4 systemctl	492
39.5 Labs	494
40 Backup and Recovery Methods	495

For the exclusive use of LFS301 corp class taught 06 to 09 December

40.1	Backup Basics	496
40.2	Backup vs Archive	498
40.3	Backup Methods and Strategies	500
40.4	tar	503
40.5	Compression: gzip, bzip2 and xz and Backups	506
40.6	dd	507
40.7	rsync	508
40.8	cpio **	509
40.9	Backup Programs **	510
40.10	Labs	511
41	Linux Security Modules	515
41.1	Linux Security Modules	516
41.2	SELinux	518
41.3	AppArmor	530
41.4	Labs	534
42	Local System Security	539
42.1	Local System Security	540
42.2	Creating a Security Policy	541
42.3	Updates and Security	545
42.4	Physical Security	546
42.5	BIOS	548
42.6	Bootloader	549
42.7	Filesystem Security	550
42.8	setuid/setgid bits	551
42.9	Labs	552
43	Basic Troubleshooting	555
43.1	Troubleshooting Levels	556
43.2	Troubleshooting Techniques	557
43.3	Networking	558
43.4	File Integrity	559
43.5	Boot Process Failures	560
43.6	Filesystem Corruption and Recovery	561
43.7	Virtual Consoles	562
43.8	Labs	563
44	System Rescue	565
44.1	Rescue Media and Troubleshooting	566
44.2	Using Rescue/Recovery Media	567
44.3	System Rescue and Recovery	568
44.4	Emergency Boot Media	569
44.5	Using Rescue Media	570
44.6	Emergency Mode	572
44.7	Single User Mode	573
44.8	Labs	574
45	Closing and Evaluation Survey	577
45.1	Evaluation Survey	577

**Please Note**

** These sections may be considered in part or in whole as optional. They contain either background reference material, specialized topics, or advanced subjects. The instructor may choose to cover or not cover them depending on

For the exclusive use of LFS301 corp class taught 06 to 09 December



classroom experience and time constraints.

For the exclusive use of LFS301 corp class taught 06 to 09 December

List of Figures

1.1	Linux Foundation Digital Badges	10
2.1	/bin Directory: Ubuntu 18.04	31
2.2	/boot Directory on Ubuntu 20.04	32
2.3	Device Nodes	33
2.4	/dev Directory	33
2.5	/home Directory	36
2.6	/proc Directory	42
2.7	/proc/[pid] Directory	42
2.8	Contents of /proc/interrupts	42
2.9	/sys Directory	43
2.10	/root Directory	44
2.11	/sbin Directory: Ubuntu 18.04	46
2.12	/var Directory	50
2.13	/run Directory	51
3.1	ulimit	60
3.2	User and Kernel Modes and System Calls	67
4.1	Available Signals	81
5.1	RPM and APT	93
6.1	Uninstalling RPM Packages	110
7.1	Getting Source Packages on Ubuntu	119
8.1	rpm Repositories	125
11.1	Using sar	152
11.2	Using sar to Get I/O Statistics	153
12.1	Using ps With BSD Options	161
12.2	Customizing ps Output	162
12.3	Using ps With UNIX Options	162
12.4	Using top	164
12.5	/proc Contents	165
13.1	/proc/sys/vm	172
13.2	vmstat	173
13.3	Using vmstat	173
13.4	Using vmstat on One Disk	173
14.1	iostat	179
14.2	Using iostat	179
14.3	Using iotop	180
14.4	iotop options	180

For the exclusive use of LFS301 corp class taught 06 to 09 December

16.1 Hard and Soft Links	194
17.1 Using fdisk	205
17.2 MBR Disk Partition Table	208
17.3 GPT Disk Partition Table	209
17.4 Using blkid	211
17.5 Using lsblk	212
18.1 mkfs	225
18.2 fsck	226
18.3 mount	228
18.4 Currently Mounted Filesystems	229
18.5 /etc/fstab Example	233
18.6 automount Example	235
19.1 Using df	242
19.2 Using du	243
20.1 ext[3,4] Filesystem Layout	257
20.2 Block Groups	258
23.1 LVM components	283
23.2 Logical Volume Utilities	284
23.3 Physical Volume Utilities	284
23.4 Volume Group Utilities	286
25.1 sysctl	303
26.1 Using lsmod	309
26.2 modinfo	312
27.1 Device Nodes	317
27.2 udev Rules	319
28.1 Emulators	329
28.2 Dedicated Hypervisor	333
28.3 Hypervisor in the Kernel	334
28.4 libvirt-based Utilities	336
28.5 Starting virt-manager	343
28.6 Creating a Virtual Machine with virt-manager	344
28.7 Selecting the TinyCoreLinux iso image in virt-manager	344
28.8 Configuring Memory and CPUS in virt-manager	345
28.9 Configuring Disk Storage in virt-manager	345
28.10 Beginning the VM installation in virt-manager	346
28.11 Adding an Input Device to the VM in virt-manager	347
28.12 Booting into the Installation Media with virt-manager	347
28.13 First TinyCoreLinux Screen	348
28.14 First TinyCoreLinux Screen Resized	348
28.15 Running tc-install	349
28.16 Selecting Disk in VM	349
28.17 Finishing Installation in virt-manager	350
28.18 Running the new VM in virt-manager	350
29.1 Containers	354
29.2 Application Virtualization	355
29.3 Checking docker status	361
29.4 Using docker search	362
30.1 Using usermod	369

For the exclusive use of LFS301 corp class taught 06 to 09 December

30.2	Using chage	375
32.1	Using chmod	397
33.1	PAM Configuration Files	407
34.1	IP Addresses	412
35.1	Using ip	424
35.2	ifconfig Output	425
35.3	nmtui Main Screen	431
35.4	nmtui Edit Screen	431
35.5	nmtui Wireless Configuration	432
35.6	nmcli	433
35.7	Using route and ip route	434
35.8	DNS	439
35.9	ping	441
35.10	traceroute	442
35.11	mtr	443
37.1	Boot Sequence	468
37.2	Configuration Example: /etc/sysconfig/selinux	472
37.3	/etc/sysconfig on RHEL 8	472
37.4	/etc/default	473
37.5	shutdown Command	474
38.1	/etc/default/grub on RHEL 8	483
38.2	/etc/grub.d contents	483
41.1	SELinux Enforcing Mode	519
41.2	SELinux Permissive Mode	519
41.3	semanage	528
45.1	Course Survey	577

For the exclusive use of LFS301 corp class taught 06 to 09 December

For the exclusive use of LFS301 corp class taught 06 to 09 December

List of Tables

2.1	Main Directories Part I	27
2.2	Main Directories Part II	28
2.3	Directories Under /usr	49
2.4	Directories Under /var	50
5.1	Available Source Control Systems	98
6.1	rpm Query Command Examples	107
11.1	Network Monitoring Utilities	151
11.2	sar Options	153
11.3	System Log Files	155
12.1	Process and Load Monitoring Tools	160
13.1	Memory Monitoring Utilities	170
14.1	I/O Scheduling Classes	181
16.1	Special Filesystems	199
35.1	ip	423
41.1	AppArmor Utilities	533

For the exclusive use of LFS301 corp class taught 06 to 09 December

Chapter 1

Introduction



1.1	Linux Foundation	2
1.2	Linux Foundation Training	4
1.3	Linux Foundation Certifications	7
1.4	Linux Foundation Digital Badges	10
1.5	Laboratory Exercises, Solutions and Resources	11
1.6	E-Learning Course: LFS201	13
1.7	Distribution Details	14
1.8	Labs	21

For the exclusive use of LFS301 corp class taught 06 to 09 December

1.1 Linux Foundation

What is the Linux Foundation?

- A non-profit consortium, dedicated to fostering the growth of:
 - **Linux**
 - Many other **Open Source Software (OSS)** projects and communities
- Supports the creation of sustainable **OSS** ecosystems by providing:
 - Financial and intellectual resources and services
 - Training
 - Events
- Originally founded to protect, support and improve **Linux** development and sponsors the work of **Linux** creator Linus Torvalds
- Supported by leading technology companies and developers in a neutral collaborative environment
- See <https://www.linuxfoundation.org>.

The **Linux Foundation** provides a neutral, trusted hub for developers to code, manage, and scale open technology projects. Founded in 2000, The **Linux Foundation** is supported by more than 1,000 members and is the world's leading home for collaboration on open source software, open standards, open data and open hardware. The **Linux Foundation**'s methodology focuses on leveraging best practices and addressing the needs of contributors, users and solution providers to create sustainable models for open collaboration.

The **Linux Foundation** hosts **Linux**, the world's largest and most pervasive open source software project in history. It is also home to **Linux** creator Linus Torvalds and lead maintainer Greg Kroah-Hartman. The success of **Linux** has catalyzed growth in the open source community, demonstrating the commercial efficacy of open source and inspiring countless new projects across all industries and levels of the technology stack.

As a result, the **Linux Foundation** today hosts far more than **Linux**; it is the umbrella for many critical open source projects that power corporations today, spanning virtually all industry sectors. Some of the technologies we focus on include big data and analytics, networking, embedded systems and IoT, web tools, cloud computing, edge computing, automotive, security, blockchain, and many more.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Linux Foundation Events

This is only a very partial list of **Linux Foundation** events. Some are held in multiple locations yearly, such as North America, Europe and Asia.

- Open Source Summit
- Embedded Linux Conference North
- Open Networking & Edge Summit
- KubeCon + CloudNativeCon
- Automotive Linux Summit
- KVM Forum
- Linux Storage Filesystem and Memory Management Summit
- Linux Security Summit
- Linux Kernel Maintainer Summit
- The Linux Foundation Member Summit
- Open Compliance Summit
- And many more.

Over 85,000 open source technologists and leaders worldwide gather at **Linux Foundation** events annually to share ideas, learn and collaborate. **Linux Foundation** events are the meeting place of choice for open source maintainers, developers, architects, infrastructure managers, and sysadmins and technologists leading open source program offices, and other critical leadership functions.

These events are the best place to gain visibility within the open source community quickly and advance open source development work by forming connections with the people evaluating and creating the next generation of technology. They provide a forum to share and gain knowledge, help organizations identify software trends early to inform future technology investments, connect employers with talent, and showcase technologies and services to influential open source professionals, media, and analysts around the globe.

For the exclusive use of LFS301 corp class taught 06 to 09 December

1.2 Linux Foundation Training

Training Venues

The **Linux Foundation** offers several types of training:

- Physical Classroom (often On-Site)
- Online Virtual Classroom
- Individual Self-Paced E-learning over the Internet
- Events-Based

The **Linux Foundation**'s training is for the community, by the community, and features instructors and content straight from the leaders of the developer community.

Attendees receive training that is operating system and/or **Linux** distribution-flexible, technically advanced and created with the actual leaders of the development community themselves. **Linux Foundation** courses give attendees the broad, foundational knowledge and networking needed to thrive in their careers today. With either online or in person training, the **Linux Foundation** classes can keep you or your developers ahead of the curve on the essentials of open source administration and development.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Training Offerings

Our current course offerings include:

- Linux Programming & Development Training
- Enterprise IT & Linux System Administration Courses
- Open Source Compliance Courses

For more information see <https://training.linuxfoundation.org>.

The **Linux Foundation** also offers a wide range of free **MOOCs** (**M**assively **O**pen **O**nline **C**ourses) offered through **edX** at <https://www.edx.org>. These cover basic as well as rather advanced topics associated with open source.

To find them at the **edX** website, search on "Linux Foundation"

For the exclusive use of LFS301 corp class taught 06 to 09 December

Copyright

- The contents of this course and all its related materials, including hand-outs, are © Copyright the Linux Foundation 2021. All rights reserved.



Do not copy or distribute

No part of this publication may be stored in a retrieval system, transmitted or reproduced in any way, including, but not limited to, photocopy, photograph, magnetic, electronic or other record, without the prior written permission of The Linux Foundation.

This training, including all material provided herein, is supplied without any guarantees from **The Linux Foundation**. **The Linux Foundation** assumes no liability for damages or legal action arising from the use or misuse of contents or details contained herein.

If you believe **The Linux Foundation** materials are being used, copied, or otherwise improperly distributed please email info@linuxfoundation.org.

For the exclusive use of LFS301 corp class taught 06 to 09 December

1.3 Linux Foundation Certifications

LFCS and LFCE



The **Linux Foundation** offers a two-level **Certification Program**:

- **LFCS** (**L**inux **F**oundation **C**ertified **S**ysadmin)
- **LFCE** (**L**inux **F**oundation **C**ertified **E**ngineer)

Full details about this program can be found at <https://training.linuxfoundation.org/certification>.

This information includes a thorough description of the **Domains** and **Competencies** covered by each exam.

Besides the **LFCS** and **LFCE** exams, the **Linux Foundation** is currently associated with other open source certification programs involving its collaborative projects, for which it has created (or is currently creating) the exams. These include:

- Certified Kubernetes Administrator (CKA)
- Certified Kubernetes Application Developer (CKAD)
- Certified Kubernetes Application Security Specialist (CKAS)
- Linux Foundation Certified IT Associate (LFCA)
- FinOps Certified Practitioner (FOCP)
- Certified Hyperledger Fabric Administrator (CHFA)
- Certified Hyperledger Sawtooth Administrator (CHSA)
- OpenJS Node.js Services Developer (JSNSD)
- OpenJS Node.js Application Developer (JSNAD)
- Cloud Foundry Certified Developer (CFCD)
- Certified ONAP Professional (COP)

For the exclusive use of LFS301 corp class taught 06 to 09 December

Certification/Training Firewall

- The **Linux Foundation** has two separate training divisions:
 - Certification
 - Course Delivery
- These are separated by a **firewall**:
 - Enables third party organizations to develop and deliver **LF** certification preparation classes
 - Prevents using **secret sauce** in **LF** courses
 - Prevents **teaching the test** in **LF** courses
- Instructors (including today) are guided entirely by publicly available information

The curriculum development and maintenance division of the **Linux Foundation** training department has no direct role in developing, administering, or grading certification exams.

Enforcing this self-imposed **firewall** ensures that independent organizations and companies can develop third party training material, geared to helping test takers pass their certification exams.

Furthermore, it ensures that there are no secret “tips” (or secrets in general) that one needs to be familiar with to succeed.

It also permits the **Linux Foundation** to develop a very robust set of courses that do far more than **teach the test**, but rather equip attendees with a broad knowledge of many areas they may be required to master to have a successful career in **Linux** system administration.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Preparation Resources

- Before doing anything else, download:
<https://training.linuxfoundation.org/download-free-certification-prep-guide>
- Sections on:
 - Domains and Competencies
 - Free Training Resources
 - Paid Training Resources
 - Taking the Exam
- Also get the **Candidate Handbook** at:

https://training.linuxfoundation.org/go/candidate_handbook

- Also read exam-specific information at:

<https://training.linuxfoundation.org/certification/lfcs>

<https://training.linuxfoundation.org/certification/lfce>

We will discuss some (but not all!) of the issues in the documents quoted above, all of which can also be easily be found through links at the **Linux Foundation** web page at <https://training.linuxfoundation.org/certification>.

Topics covered include:

- What material is covered in the exam
- Candidate Requirements, including identification, authentication, eligibility, accessibility, confidentiality requirements.
- Exam registration and fees, refund policy, etc.
- **Linux** distribution choices
- Checking your hardware and software environment for suitability for the exam
- How to start and complete the exam
- Exam Interface and format
- Exam results, scoring and re-scoring requests, and retake policy
- Certificate issuance, verification, expiration, renewal etc.
- Accessing tech support

For the exclusive use of LFS301 corp class taught 06 to 09 December

1.4 Linux Foundation Digital Badges



Figure 1.1: Linux Foundation Digital Badges

- Can be used in email signatures, digital resumes, social media sites (**LinkedIn**, **Facebook**, **Twitter**, etc)
- Contain verified metadata describing qualifications and process that earned them.
- Available to students who successfully complete **Linux Foundation** courses and certifications
- Details at <https://training.linuxfoundation.org/badges/>

- Digital Badges communicate abilities and credentials

The **Linux Foundation** is committed to providing you with the tools necessary to achieve your professional goals. We understand communicating your abilities and credentials can be challenging. For this reason, we have partnered with **Credly** to provide you with a digital version of your credentials through its **Acclaim** platform. These badges can be used in email signatures or digital resumes, as well as on social media sites such as **LinkedIn**, **Facebook**, and **Twitter**. This digital image contains verified metadata that describes your qualifications and the process required to earn them.

- Badges are shareable via any digital platform: social media, embedded in your résumé, email signature, or the web.
- All badges can be verified by anyone simply by clicking on the badge.
- Badges will be issued to everyone who passes one of our certification exams as well as those who purchase training courses directly from the Linux Foundation or an authorized training partner.
- Badges will also be issued to those who contributed on our exam or course development teams

How it Works

1. You will receive an email notifying you to claim your badge at our partner **Credly's Acclaim** platform website.
2. Click the link in that email.
3. Create an account on the **Acclaim** platform site and confirm your email.
4. Claim your badge.
5. Start sharing.

For the exclusive use of LFS301 corp class taught 06 to 09 December

1.5 Laboratory Exercises, Solutions and Resources

Labs

- Hands-on exercises provided at the end of each session.
- Can be done on either virtual machines or bare-metal
 - Unless otherwise specified
- Some exercises marked as optional
- Solutions available at the end of each exercise.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Obtaining Course Solutions and Resources

- If this course has such material, lab exercise solutions, suggestions and other resources may be downloaded from: <https://training.linuxfoundation.org/cm/LFS301>

- If you do not have a browser available you can obtain with:

```
$ wget --user=LFtraining --password=<ask instructor> \
  https://training.linuxfoundation.org/cm/LFS301/LFS301_V7.2_SOLUTIONS.tar.xz
```

and similarly for the RESOURCES file.

```
$ wget --user=LFtraining --password=<ask instructor> \
  https://training.linuxfoundation.org/cm/LFS301/LFS301_V7.2_RESOURCES.tar
```

- Any errata, updated solutions, etc. will also be posted on that site
- These files may be unpacked with:

```
$ tar xvf LFS301_V7.2_SOLUTIONS.tar.xz
$ tar xvf LFS301_V7.2_RESOURCES.tar
```

You will see subdirectories in both the **SOLUTIONS** and **RESOURCES** directories such as s_01, s_10, s_13 for each section that has files you either need or may find of interest. We may refer to these files in future sections.

Depending on the course, there may not be a **RESOURCES** file, which is intended for binary files such as archives and videos.

Binary files, such as source tarballs for various labs, will be resourced with instructions as needed.

If the course has demonstration videos included, these are mostly intended for supplementary study outside of lecture time. However, the instruction might elect to show some of them to get a hopefully clean demonstration, or give an example of practices on alternative **Linux** distributions.

For the exclusive use of LFS301 corp class taught 06 to 09 December

1.6 E-Learning Course: LFS201

Companion E-Learning Course: LFS201

- The **Linux Foundation** offers a self-paced, e-learning closely related course:
LFS201: Essentials of Linux System Administration
- This course covers much of the same material and shares many of the laboratory exercises as this course.
- Access to materials online is available for one year and you can take as much time as you need to cover all the content and exercises.
- Note that this instructor-led version cannot give adequate time to all subjects either in breadth or depth due to time constraints, and some topics are marked as optional.
- You may have received a subscription to **LFS201** as part of your enrollment in this course. Otherwise you can contact separately through <https://training.linuxfoundation.org/>.

For the exclusive use of LFS301 corp class taught 06 to 09 December

1.7 Distribution Details

Software Environment

- Class material is designed for multiple environments
- Focuses on three current major **Linux** distribution families:
 - **Red Hat / Fedora**
 - **OpenSUSE / SUSE**
 - **Debian**



Please Note

The material in this section is aimed at courses which either require or strongly recommend they be run only on a **Linux**-based operating system. Some courses can be run using any environment that has a browser and perhaps an **ssh** (secure shell) utility. However, while in that case you can skip over this material, it is still recommended you absorb its information.

The material produced by the **Linux Foundation** is distribution-flexible. This means that technical explanations, labs and procedures should work on most modern distributions and we do not promote products sold by any specific vendor (although we may mention them for specific scenarios).

In practice, most of our material is written with the three main **Linux** distribution families in mind: **Red Hat / Fedora**, **OpenSUSE / SUSE** and **Debian**. Distributions used by our students tend to be one of those three alternatives, or a product that's derived from them.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Which Distribution to Choose

- Several factors to consider:
 - Has your employer already standardized?
 - Do you want to learn more?
 - Do you want to certify?
- You are encouraged to experiment with more than one distribution

You should ask yourself several questions when choosing a new distribution. While there are many reasons that may force you to focus on one **Linux** distribution versus another, we encourage you to gain experience on all of them. You will quickly notice that technical differences are mainly about package management systems, software versions and file locations. Once you get a grasp of those differences it becomes relatively painless to switch from one **Linux** distribution to another.

Some tools and utilities have vendor supplied front-ends, especially for more particular or complex reporting. The steps included in the text may need to be modified to run on a different platform.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Red Hat / Fedora Family

- Current material based upon the latest releases of **Red Hat Enterprise Linux (RHEL)**.
- Supports **x86**, **x86-64**, **Itanium**, **PowerPC** and **IBM System Z**
- RPM-based, uses **dnf** (or **yum**) to install and update
- Long release cycle; targets enterprise server environments
- Upstream for **CentOS** and **Oracle Linux**
- Downstream for **CentOS Stream**



CentOS Stream

CentOS Stream is used for demos and labs because it is available at no cost.

Fedora is the community distribution that forms the basis of **Red Hat Enterprise Linux**, **CentOS**, and **Oracle Linux**. **Fedora** contains significantly more software than **Red Hat's** enterprise version. One reason for this is a diverse community is involved in building **Fedora**; it is not just one company.

The **Fedora** community produces new versions every six months or so. For this reason, we decided to standardize the **Red Hat / Fedora** part of the course material on the latest version of **CentOS/CentOS Stream**, which provides much longer release cycles. Once installed, **CentOS Stream** is also very close to **Red Hat Enterprise Linux (RHEL)**, which is the most popular **Linux** distribution in enterprise environments.



CentOS and CentOS Stream

- **CentOS** historically has been basically a copy of **RHEL** with some time delay after updates.
- **CentOS Stream** gets updates **before RHEL**, but otherwise is quite close to it. Thus newer features will be absorbed quicker.
- **Red Hat** is ending support for **CentOS 8** at the end of 2021. Thus, this course is now tested with the **CentOS Stream** distribution; any variance from **CentOS 8** or **RHEL 8** will be minor and should not even be noticeable.

For the exclusive use of LFS301 corp class taught 06 to 09 December

OpenSUSE Family

- Current material based upon the latest release of **OpenSUSE** and should work well with later versions.
- RPM-based, uses **zypper** to install and update.
- **YaST** available for administration purposes
- **x86** and **x86-64**
- Upstream for **SUSE Linux Enterprise Server (SLES)**



Please Note

OpenSUSE is used for demos and labs because it is available at no cost.

The relationship between **OpenSUSE** and **SUSE Linux Enterprise Server** is similar to the one we just described between **Fedora** and **Red Hat Enterprise Linux**. In this case, however, we decided to use **OpenSUSE** as the reference distribution for the **OpenSUSE** family due to the difficulty of obtaining a free version of **SUSE Linux Enterprise Server**. The two products are extremely similar and material that covers **OpenSUSE** can typically be applied to **SUSE Linux Enterprise Server** with no problem.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Debian Family

- Commonly used on both servers and desktop
- **DPKG**-based, use **apt** and front-ends for installing and updating
- Upstream for **Ubuntu**, **Linux Mint** and others
- Current material based upon the latest release of **Ubuntu** and should work well with later versions.
- **x86** and **x86-64**
 - Long Term Release (LTS)



Please Note

Ubuntu is used for demos and labs because it is available at no cost, as is **Debian**, but has a wider base with new **Linux** users.

The **Debian** distribution is the upstream for several other distributions including **Ubuntu**, **Linux Mint** and others. **Debian** is a pure open source project and focuses on a key aspect: stability. It also provides the largest and most complete software repository to its users.

Ubuntu aims at providing a good compromise between long term stability and ease of use. Since **Ubuntu** gets most of its packages from **Debian**'s unstable branch, **Ubuntu** also has access to a very large software repository. For those reasons, we decided to use **Ubuntu** as the reference **Debian**-based distribution for our lab exercises.

For the exclusive use of LFS301 corp class taught 06 to 09 December

New Distribution Similarities

- Current trends and changes to the distributions have reduced some of the differences between the distributions.
 - **systemd**, system startup and service management
 - **journald**, manage system logs
 - **firewalld**, firewall management daemon
 - **ip**, network display and configuration tool



Please Note

Since these utilities are common across distributions, the lecture and lab information will be mostly based on them.

If your choice of distribution or release does not support these commands, please translate accordingly.

systemd is used by the most common distributions replacing the **SysVinit** and **Upstart** packages. Replaces **service** and **chkconfig** commands.

journald is a **systemd** service that collects and stores logging data. It creates and maintains structured, indexed journals based on logging information that is received from a variety of sources. Depending on the distribution text based system logs may be replaced.

firewalld provides a dynamically managed firewall with support for network/firewall zones to define the trust level of network connections or interfaces. It has support for IPv4, IPv6 firewall settings and for ethernet bridges. This replaces the **iptables** configurations.

The **ip** program is part of the **net-tools** package and is designed to be a replacement for the **ifconfig** command. The **ip** command will show or manipulate routing, network devices, routing information and tunnels.

These documents may be of some assistance translating older commands to their **systemd** counterparts:

https://fedoraproject.org/wiki/SysVinit_to_Systemd_Cheatsheet

<https://wiki.debian.org/systemd/CheatSheet>

https://en.opensuse.org/openSUSE:Cheat_sheet_13.1#Services

For the exclusive use of LFS301 corp class taught 06 to 09 December

AWS Free Tier

- **Amazon Web Services (AWS)** offers a wide range of virtual machine products (**instances**) which remote users can access in the cloud.
- In particular, one can use their **Free Tier** account level for up to a year and the simulated hardware and software choices available may be all one needs to perform the exercises for **Linux Foundation** training courses and gain experience with **Linux**. Or they may furnish a very educational supplement to working on local hardware, and offer opportunities to easily study more than one **Linux** distribution. Please download our guide to help you experiment with the **AWS** free tier from <https://training.linuxfoundation.org/cm/prep/aws.pdf>.
- Note that **AWS** will not give access to the console and will not present a Graphical interface, so you will not be able to perform tasks which require either of these facilities. Furthermore, for kernel-level courses there are some particular challenges.

For the exclusive use of LFS301 corp class taught 06 to 09 December

1.8 Labs

Exercise 1.1: Configuring the System for sudo

It is very dangerous to run a **root shell** unless absolutely necessary: a single typo or other mistake can cause serious (even fatal) damage.

Thus, the sensible procedure is to configure things such that single commands may be run with superuser privilege, by using the **sudo** mechanism. With **sudo** the user only needs to know their own password and never needs to know the root password.

If you are using a distribution such as **Ubuntu**, you may not need to do this lab to get **sudo** configured properly for the course. However, you should still make sure you understand the procedure.

To check if your system is already configured to let the user account you are using run **sudo**, just do a simple command like:

```
$ sudo ls
```

You should be prompted for your user password and then the command should execute. If instead, you get an error message you need to execute the following procedure.

Launch a root shell by typing **su** and then giving the **root** password, not your user password.

On all recent **Linux** distributions you should navigate to the `/etc/sudoers.d` subdirectory and create a file, usually with the name of the user to whom root wishes to grant **sudo** access. However, this convention is not actually necessary as **sudo** will scan all files in this directory as needed. The file can simply contain:

```
student ALL=(ALL)  ALL
```

if the user is **student**.

An older practice (which certainly still works) is to add such a line at the end of the file `/etc/sudoers`. It is best to do so using the **visudo** program, which is careful about making sure you use the right syntax in your edit.

You probably also need to set proper permissions on the file by typing:

```
$ sudo chmod 440 /etc/sudoers.d/student
```

(Note some **Linux** distributions may require 400 instead of 440 for the permissions.)

After you have done these steps, exit the root shell by typing **exit** and then try to do **sudo ls** again.

There are many other ways an administrator can configure **sudo**, including specifying only certain permissions for certain users, limiting searched paths etc. The `/etc/sudoers` file is very well self-documented.

However, there is one more setting we highly recommend you do, even if your system already has **sudo** configured. Most distributions establish a different path for finding executables for normal users as compared to root users. In particular the directories `/sbin` and `/usr/sbin` are not searched, since **sudo** inherits the `PATH` of the user, not the full root user.

Thus, in this course we would have to be constantly reminding you of the full path to many system administration utilities; any enhancement to security is probably not worth the extra typing and figuring out which directories these programs are in. Consequently, we suggest you add the following line to the `.bashrc` file in your home directory:

```
PATH=$PATH:/usr/sbin:/sbin
```

If you log out and then log in again (you don't have to reboot) this will be fully effective.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Chapter 2

Linux Filesystem Tree Layout



2.1	One Big Filesystem	24
2.2	Data Distinctions	25
2.3	FHS Linux Standard Directory Tree	26
2.4	root (/) directory	29
2.5	/bin	30
2.6	/boot	32
2.7	/dev	33
2.8	/etc	34
2.9	/home	35
2.10	/lib and /lib64	37
2.11	/media	38
2.12	/mnt	39
2.13	/opt	40
2.14	/proc	41
2.15	/sys	43
2.16	/root	44
2.17	/sbin	45
2.18	/srv	47
2.19	/tmp	48
2.20	/usr	49
2.21	/var	50
2.22	/run	51
2.23	Labs	52

2.1 One Big Filesystem

One Big Filesystem

- Seen as one big filesystem
- Inverted tree with `/` root directory at the top
- Additional partitions and filesystems can be **mounted** on subdirectories
- Standardized prescriptions followed by **Linux** distributions about what goes where

Linux, like all **UNIX**-based operating systems, consists of one big filesystem tree. Actually, it is usually diagrammed as an inverted tree with the root directory, `/`, being at the top of the tree.

Within this one large logical filesystem there may be more than one, even many, distinct filesystems, **mounted** at various points, which appear as subdirectories. These distinct filesystems are usually on different partitions, which can be on any number of devices, including those which are on a network.

Regardless of exactly how things are joined together, it all just looks like one big filesystem; applications do not usually care at all about what physical device files actually reside on.

Once upon a time, different **UNIX**-like operating systems organized this one big tree in varying ways; even among the various **Linux** distributions there were many differences. This made both writing applications and accomplishing system administration tasks on more than one kind of system difficult and often frustrating.

As a consequence, the **Linux** ecosystem has worked hard to establish standardized procedures to minimize such pain.

For the exclusive use of LFS301 corp class taught 06 to 09 December

2.2 Data Distinctions

Data Distinctions

- Shareable vs non-shareable
- Variable vs static

When talking about how files and data are organized in the one big directory tree, it is important to learn some taxonomy on what kind of information has to be read and written. In particular, there are two kinds of distinctions:

1. Shareable vs. non-shareable

Shareable data is that which can be shared between different hosts. Non-shareable is that which must be specific to a particular host. For example, user home directories may be shareable while device lock files are not.

2. Variable vs. static

Static data include binaries, libraries, documentation, and anything that does not change without system administrator assistance. Variable data is anything that may change even without a system administrator's help.

Often these logical distinctions are embodied as different kinds of information residing in various directories, or even partitions and filesystems.

For the exclusive use of LFS301 corp class taught 06 to 09 December

2.3 FHS Linux Standard Directory Tree

The Filesystem Hierarchy Standard

- Standard layout of types and locations of files.
 - Makes prediction of file locations possible
- Grown out of the historical standards from BSD and others.
- Administered under the **Linux Foundation**: https://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf.

The **Filesystem Hierarchy Standard (FHS)**, administered originally by the **Free Standards Group**, and now by the **Linux Foundation**, specifies the main directories that need to be present and describes their purposes. It can be downloaded from https://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf.

By specifying a standard layout the **FHS** simplifies predictions of file locations. While most **Linux** distributions respect the **FHS**, probably none of them follow it exactly, and the last official version is old enough that it does not take into account some new developments.

Distributions like to experiment and eventually some of the experiments become generally accepted.

For the exclusive use of LFS301 corp class taught 06 to 09 December

FHS Linux Standard Directory Tree

Table 2.1: Main Directories Part I

Directory Purpose	
/	Primary directory of the entire file system hierarchy.
/bin	Essential executable programs that must be available in single user mode .
/boot	Files needed to boot the system, such as the kernel, initrd or initramfs images, and boot configuration files and bootloader programs.
/dev	Device Nodes , used to interact with hardware and software devices.
/etc	System wide configuration files.
/home	User home directories including personal settings, files, etc.
/lib	Libraries required by executable binaries in /bin and /sbin.
/lib64	64-bit Libraries required by executable binaries in /bin and /sbin, for systems which can run both 32-bit and 64-bit programs.
/media	Mount points for removable media such as CDs , DVDs , USB sticks etc.
/mnt	Temporarily mounted filesystems.

There may be additional distribution-specific directories found under the root directory. These might include `/misc` which can be used for miscellaneous data, and `/tftpboot`, which is used for booting using **tftp**. If there are files in the directory, they are related to diskless workstation booting. Note it does not violate the **FHS** to have other directories; however, it does violate it to have components in directories other than those dictated by the standard.

FHS Linux Standard Directory Tree, Continued

Table 2.2: Main Directories Part II

Directory	Purpose
<code>/opt</code>	Optional application software packages.
<code>/proc</code>	Virtual pseudo-filesystem giving information about the system and processes running on it. Can be used to alter system parameters.
<code>/run</code>	Run-time variable data, containing information describing the system since it was booted. Replaces the older <code>/var/run</code> .
<code>/sys</code>	Virtual pseudo-filesystem giving information about the system and processes running on it. Can be used to alter system parameters. Similar to a device tree and is part of the Unified Device Model .
<code>/root</code>	Home directory for the root user.
<code>/sbin</code>	Essential system binaries.
<code>/srv</code>	Site-specific data served up by the system. Seldom used.
<code>/tmp</code>	Temporary files; on many distributions lost across a reboot and may be a ramdisk in memory.
<code>/usr</code>	Multi-user applications, utilities and data; theoretically read-only.
<code>/var</code>	Variable data that changes during system operation.

These are additional directories off the main **root** (/) directory.

For the exclusive use of LFS301 corp class taught 06 to 09 December

2.4 root (/) directory

root (/) directory

- Must be adequate to
 - Boot
 - Restore
 - Recover
 - Repair
 the system as needed
- No application or package should create new subdirectories of the root directory



/ vs /root

We are not talking about `/root`, which is the home directory of **root**, the **superuser**.

While the entire filesystem can be viewed as one big tree, as we have pointed out, there may be multiple partitions and filesystems joined together.

The partition and filesystem that the root directory itself is contained in is rather special and is often in a special dedicated partition, with other components with directories such as `/home`, `/var` and `/opt` etc. to be mounted later.

The root partition must contain all essential files required to boot the system and then mount all other filesystems. Thus, it needs utilities, configuration files, boot loader information, and other essential startup data. It must be adequate to:

- Boot the system.
- Restore the system from system backups on external media such as tapes and other removable media or **NAS** etc.
- Recover and/or repair the system; an experienced maintainer must have the tools to diagnose and reconstruct a damaged system.

According to the **FHS**, no application or package should create new subdirectories of the root directory.



/ is not /root

`/root` is the home directory of the superuser. The word choices here can be confusing!

For the exclusive use of LFS301 corp class taught 06 to 09 December

2.5 /bin

/bin

- Contains executable programs and scripts needed by both system administrators and unprivileged users, which are required when no other filesystems have yet been mounted, for example when booting into **single user** or recovery mode.
- May also contain executables which are used indirectly by scripts.
- May not include any subdirectories.

/bin and /usr/bin

Some recent distributions have abandoned the strategy of separating `/bin` and `/usr/bin` (as well as `/sbin` and `/usr/sbin`) and just have one directory with symbolic links, thereby preserving a two directory view. They view the time-honored concept of enabling the possibility of placing `/usr` on a separate partition to be mounted after boot as obsolete.

Required programs which must exist in `/bin` include:

`cat, chgrp, chmod, chown, cp, date, dd, df, dmesg, echo, false, hostname, kill, ln, login, ls, mkdir, mknod, more, mount, mv, ps, pwd, rm, rmdir, sed, sh, stty, su, sync, true, umount and uname.`

[and `test` may be there as well, and optionally, it may include:

`csh, ed, tar, cpio, gzip, zcat, netstat and ping.`

Command binaries that are deemed not essential enough to merit a place in `/bin` go in `/usr/bin`. Programs required only by non-root users are placed in this category.

For the exclusive use of LFS301 corp class taught 06 to 09 December

/bin Example

```

student@ubuntu:~$ ls /bin
bzgrep      eftbootmgr  lesskey   networkctl  pwd       systemd-machine-id-setup    zforce
bztp2       egrep       lesspipe  ln           readlink  systemd-notify          zgrep
bztp2recover false      loadkeys  ntfs-3g     red       systemd-sysusers        zless
bzless      fgconsole   login    ntfs-3g.probe rm       systemd-tmpfiles       zmore
bzmore     fgrep       ntfscluster  ntfscluster rnano   systemd-tty-ask-password-agent znew
cat        findmnt   logindctl  ntfscluster  rnano   tar
student@ubuntu:~$ ls /bin
bash        chacl      fsck.btrfs  lowntfs-3g  ntfscmp   run-parts      tempfile
brltty     chgrp      fuser      ls          ntfsfallocate sed          touch
btrfs      chmod      fusermount lsblk      ntfsvlx   setfacl       true
btrfsck    chown      getfacl   lsmod      ntfsiinfo  setfont       udevadm
btrfs-debug-tree chvt      grep      mkdir      ntfsls    setupcon      unlockngr_server
btrfs-find-root cp       gunzip   nmkfs.btrfs  ntfsmove  sh           umount
btrfs-image cpio       gzexe    nknod      ntsrecover sh.distrib  uname
btrfs-map-logical dash      gzip     nktemp    ntssecaudit sleep       uncompress
btrfs-select-super date      hciconfig more      ntfstruncate ss           uncode_start
btrfsstune dd       hostname  mount      ntfsvusermap static-sh  vdir
btrfs-zero-log df       ip       mountpoint  ntfswipe   stty         wdctl
bunzip2    dir       journalctl mt       open      su           which
busybox    dmesg     kbd_mode  mt-gnu    openvt    sync         whiptail
bzcat      dnsdomainname keyctl   mv       pidof     systemctl  ypdomainname
bzcmp      domainname kill     nano     ping      systemd   zcat
bzdiff    dumpkeys   kmod      nc       ping4    systemd-ask-password zcmp
bzegrep   echo       less     nc.openbsd ping6    systemd-escape  zdif
bzexec    ed       lessecho  netcat   plymouth  systemd-hwdb  zegrep
bzfgrep   eftbootdump lessfile  netstat  ps       systemd-inhibit zfgrep
bzgrep    eftbootmgr lesskey   networkctl  pwd       systemd-machine-id-setup zforce
bztp2      egrep       lesspipe  nisdomainname readlink  systemd-notify  zgrep
bztp2recover false      loadkeys  ntfs-3g     red       systemd-sysusers  zless
bzless     fgconsole   login    ntfs-3g.probe rm       systemd-tmpfiles  zmore
bzmore     fgrep       ntfscluster  ntfscluster rnano   systemd-tty-ask-password-agent znew
cat        findmnt   logindctl  ntfscluster  rnano   tar
student@ubuntu:~$ 

```

Figure 2.1: /bin Directory: Ubuntu 18.04

Recent distribution versions of **RHEL**, **CentOS**, **Fedor**a, and **Ubuntu** have symbolically linked `/bin` and `/usr/bin` so they are actually the same.

The above screenshot is from **Ubuntu 18.04**; later versions no longer have a separate un-linked directory.

2.6 /boot

/boot

- Contains everything required for the boot process. The two files which are absolutely essential are:
 - `vmlinuz`: The compressed **Linux** kernel.
 - `initramfs`: The **initial RAM Filesystem**, which is mounted before the real root filesystem becomes available.
- Stores data used before the kernel begins executing user-mode programs.
- Also includes two files used for information and debugging:
 - `config` Used to configure the kernel compilation.
 - `System.map`: Kernel **symbol table**, used for debugging
- There may be other files depending on distribution.

The exact contents of `/boot` will vary by distribution and time; on one **Ubuntu** system we have:

```
student@ubuntu:~$ ls -lF /boot
total 460556
-rw-r--r-- 1 root root 104585 Feb 15 05:47 config-5.11.0
-rw-r--r-- 1 root root 248277 Feb 23 03:16 config-5.8.0-45-generic
drwx----- 2 root root 4096 Dec 31 1969 efi/
drwxr-xr-x 4 root root 4096 Mar 16 06:43 grub/
lrwxrwxrwx 1 root root 27 Mar 16 06:40 initrd.img -> initrd.img-5.8.0-45-generic
-rw-r--r-- 1 root root 95660417 Mar 18 11:17 initrd.img-5.11.0
-rw-r--r-- 1 root root 57044061 Mar 16 06:41 initrd.img-5.8.0-45-generic
lrwxrwxrwx 1 root root 17 Mar 16 06:43 initrd.img.old -> initrd.img-5.11.0
-rw-r--r-- 1 root root 182704 Aug 18 2020 memtest86+.bin
-rw-r--r-- 1 root root 184380 Aug 18 2020 memtest86+.elf
-rw-r--r-- 1 root root 184884 Aug 18 2020 memtest86+_multiboot.bin
-rw-r--r-- 1 root root 4331637 Feb 15 05:47 System.map-5.11.0
-rw----- 1 root root 5520433 Feb 23 03:16 System.map-5.8.0-45-generic
-rw-r--r-- 1 root root 297688296 Feb 15 05:47 vmlinuz-5.11.0*
lrwxrwxrwx 1 root root 24 Mar 16 06:40 vmlinuz -> vmlinuz-5.8.0-45-generic
-rw-r--r-- 1 root root 7443632 Feb 15 05:47 vmlinuz-5.11.0
-rw----- 1 root root 9781120 Feb 24 03:43 vmlinuz-5.8.0-45-generic
lrwxrwxrwx 1 root root 14 Mar 16 06:43 vmlinuz.old -> vmlinuz-5.11.0
student@ubuntu:~$
```

Figure 2.2: /boot Directory on Ubuntu 20.04

These files have longer names which depend on the **Linux** distribution and kernel version. Also, instead of `initramfs`, one might have `initrd` (**initial ram disk**.)

For the exclusive use of LFS301 corp class taught 06 to 09 December

2.7 /dev

/dev

- Character and Block **device nodes**, or files
- Essential for proper system operation
- Modern systems manage device files automatically via **udev**
- On ancient systems (or embedded devices), can be created by **MAKEDEV** or **mknod** at install or at any other time as needed

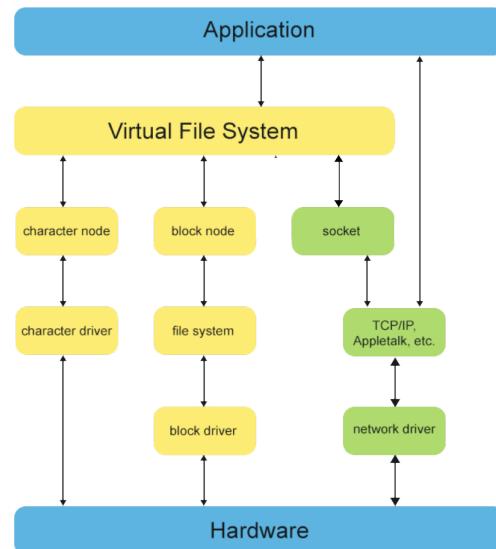


Figure 2.3: Device Nodes

This directory contains **special device files** (also known as **device nodes**) which represent devices built into or connected to the system. These special files are essential for the system to function properly. Such device files represent **character** (byte-stream) and **block** I/O devices; **network** devices do not have device nodes in **Linux** and are instead referenced by name, such as eth1 or wlan0.

```

File Edit View Search Terminal Help
c7:/tmp>ls -l /dev
total 0
...
crw----- 1 root root      5,   1 May 26 07:05 console
...
lrwxrwxrwx 1 root root      13 May 26 07:04 fd -> /proc/self/fd
...
brw-rw---- 1 root disk     7,   0 May 26 07:05 loop0
crw-rw---- 1 root disk    10, 237 May 26 07:05 loop-control
...
crw-rw---- 1 root lp       6,   0 May 26 07:05 lp0
crw-rw---- 1 root lp       6,   1 May 26 07:05 lp1
...
brw-rw---- 1 root disk     8,   0 May 26 07:05 sda
brw-rw---- 1 root disk     8,   1 May 26 07:05 sda1
brw-rw---- 1 root disk     8,   2 May 26 07:05 sda2
brw-rw---- 1 root disk     8,   3 May 26 07:05 sda3
...
lrwxrwxrwx 1 root root      15 May 26 07:04 stderr -> /proc/self/fd/2
lrwxrwxrwx 1 root root      15 May 26 07:04 stdin -> /proc/self/fd/0
lrwxrwxrwx 1 root root      15 May 26 07:04 stdout -> /proc/self/fd/1
crw-rw-rw- 1 root tty      5,   0 May 26 07:05 tty
crw-rw---- 1 root tty      4,   0 May 26 07:05 tty0
...
c7:/tmp>
c7:/tmp>
  
```

Figure 2.4: /dev Directory

All modern **Linux** distributions use the **udev** system, which creates nodes in **/dev** only as needed when devices are found. If you were to look at the **/dev** directory on an unmounted filesystem, you would find it empty.

For the exclusive use of LFS301 corp class taught 06 to 09 December

2.8 /etc

/etc

- Contains configuration files (and some startup scripts) which are local to the machine
- No binaries are allowed in this directory
- Distributions often add specific files and directories. For example, **Red Hat** extends adds a number of other directories, including `sysconfig`

This directory contains machine-local configuration files and some startup scripts; there should be no executable binary programs.

Files and directories which the **FHS** requires to be found in this directory include:

```
csh.login, exports, fstab, ftpusers, gateways, gettydefs, group, host.conf, hosts.allow,  
hosts.deny, hosts.equiv, hosts.lpd, inetd.conf, inittab, issue, ld.so.conf, motd, mtab,  
mtools.conf, networks, passwd, printcap, profile, protocols, resolv.conf, rpc, securetty,  
services, shells, syslog.conf.
```

Some of these files are pretty irrelevant today, such as `mtools.conf` which is used by floppy disks. Some will not be found any more, no matter what the **FHS** says, due to software obsolescence.

Distributions often add configuration files and directories to `/etc`. For example, **Red Hat** adds a number of other directories including `/etc/sysconfig`, where a number of system configuration files and directories live.

Other important subdirectories include:

- `/etc/skel`: Contains **skeleton** files used to populate newly created home directories).
- `/etc/systemd`: Contains or points to configuration scripts for starting, stopping, and controlling system services when using **systemd**.
- `/etc/init.d`: Contains, startup and shutdown scripts for when using **System V** initialization.

For the exclusive use of LFS301 corp class taught 06 to 09 December

2.9 /home

/home

- Usually contains the user home directories
- May be named something else
- May contain subdirectories for user groups

On **Linux** systems, user directories are conventionally placed under `/home`, as in `/home/coop`, `/home/student`, etc. All personal configuration, data, and executable programs are placed in this directory hierarchy. `/home` may also contain subdirectories for various groups or various associations of users, such as `/home/students`, `/home/staff`, `/home/aliens` etc.

On other **UNIX**-like operating systems, the concept of the `/home` directory tree exists, but can be subtly different. For example, on **Solaris** user directories are created in `/export/home` and then the **automount** facility will eventually mount them in `/home`. This is because the usual situation is that the home directory may be anywhere on a corporate network, probably on an **NFS** server, and the home directory will be mounted automatically upon use.

Linux has these same automount facilities, but many users are not even aware of them, and on self-contained systems the concept of **NFS** mounts will probably not apply.

A given user can always substitute the environmental variable `$HOME` for their root directory, or the shorthand `~`; i.e., the following are equivalent:

```
$ ls -l $HOME/public_html  
$ ls -l ~/public_html
```

There is one exception: the home directory for the **root** user on **Linux** systems is always found under `/root`. Some older **UNIX** systems may use `/` instead, which can cause clutter.

/home Example

```
student@ubuntu:~$ ls -la /home/student
total 132
drwxr-xr-x 19 student student 4096 May 26 09:36 .
drwxr-xr-x  3 root    root   4096 Apr 14 10:08 ..
-rw-----  1 student student 1494 May 25 12:45 .bash_history
-rw-r--r--  1 student student 220 Apr 14 10:08 .bash_logout
-rw-r--r--  1 student student 3771 Apr 14 10:08 .bashrc
drwx----- 14 student student 4096 May 19 12:27 .cache
drwx-----  3 student student 4096 May  1 10:07 .compiz
drwx----- 14 student student 4096 Apr 20 10:55 .config
drwx-----  3 root    root   4096 May  1 10:04 .dbus
drwxr-xr-x  2 student student 4096 Apr 14 10:13 Desktop
-rw-r--r--  1 student student 25 Apr 14 10:13 .dmrc
drwxr-xr-x  2 student student 4096 Apr 14 10:13 Documents
drwxr-xr-x  2 student student 4096 Apr 14 10:13 Downloads
drwx-----  3 root    root   4096 May  1 10:06 .emacs.d
-rw-r--r--  1 student student 8980 Apr 14 10:08 examples.desktop
drwx-----  2 student student 4096 Apr 14 10:35 .gconf
-rw-----  1 student student 4134 May 26 08:05 .ICEauthority
drwxrwxr-x  2 student student 4096 May  1 07:26 LFT
drwxrwxr-x  3 student student 4096 Apr 14 10:13 .local
drwxr-xr-x  2 student student 4096 Apr 14 10:13 Music
drwxr-xr-x  2 student student 4096 Apr 14 10:13 Pictures
-rw-r--r--  1 student student 675 Apr 14 10:08 .profile
drwxr-xr-x  2 student student 4096 Apr 14 10:13 Public
drwx-----  2 student student 4096 May 19 12:28 .ssh
-rw-r--r--  1 student student  0 Apr 14 10:15 .sudo_as_admin_successful
drwxr-xr-x  2 student student 4096 Apr 14 10:13 Templates
drwxr-xr-x  2 student student 4096 Apr 14 10:13 Videos
-rw-r--r--  1 student student 183 Apr 14 10:17 .wget-hsts
-rw-----  1 student student  51 May 26 08:05 .Xauthority
-rw-----  1 student student 3462 May 26 08:05 .xsession-errors
-rw-----  1 student student 3462 May 25 08:49 .xsession-errors.old
student@ubuntu:~$
```

Figure 2.5: /home Directory

For the exclusive use of LFS301 corp class taught 06 to 09 December

2.10 /lib and /lib64

/lib and /lib64

- Contains only those libraries which are needed to execute binaries in `/bin` and `/sbin`
- Kernel modules go in `/lib/modules`
- PAM modules go in `/lib/security`
- Some distributions put 64-bit libraries in `/lib64`

/lib and /usr/lib

Some recent distributions have abandoned the strategy of separating `/lib` and `/usr/lib` (as well as `/lib64` and `/usr/lib64`) and just have one directory with symbolic links, thereby preserving a two directory view. They view the time-honored concept of enabling the possibility of placing `/usr` on a separate partition to be mounted after boot as obsolete.

These directories should contain only those libraries needed to execute the binaries in `/bin` and `/sbin`. These libraries are particularly important for booting the system and executing commands within the root filesystem.

Kernel modules (often device or filesystem drivers) are located under `/lib/modules/(kernel-version-number)`.

PAM (Pluggable Authentication Modules) files are stored in the `/lib/security`.

Systems which support both 32-bit and 64-bit binaries need to keep both kinds of libraries on the system. On some distributions, there are separate directories for 32-bit libraries (`/lib`) and 64-bit libraries (`/lib64`.)

```
$ ls -l /lib*
1 rwxrwxrwx 1 root root 8 Apr 23 2020 /sbin -> usr/sbin
2 lwxrwxrwx 1 root root 7 Apr 23 2020 /lib -> usr/lib
3 lrwxrwxrwx 1 root root 9 Apr 23 2020 /lib64 -> usr/lib64
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

2.11 /media

/media

- Used to mount file systems on removable media
 - CD ROMS
 - DVD
 - USB drives
- May be mounted dynamically



/media vs /run

- Modern systems use `/run/media/[username]/...` instead of `/media`

This directory was typically used to mount filesystems on removable media. These include **CDs**, **DVDs**, and **USB** drives, and even Paleolithic era floppies.

Linux systems mount such media dynamically upon insertion, and **udev** creates directories and then mounts the removable filesystems there, with names that are set with **udev** rules specified in configuration files. Upon unmounting and removal, the directories used as mount points disappear.

If the media has more than one partition and filesystem, more than one entry will appear. On many **Linux** distributions the file manager (such as **Nautilus**) will pop up when the media is mounted.



/run has superseded /media

Note: With current **Linux** distributions removable media will pop up under `/run/media/[username]/...` instead of `/media`. We will discuss `/run` later.

For the exclusive use of LFS301 corp class taught 06 to 09 December

2.12 /mnt

/mnt

- Special place in the file system for mounting other file systems temporarily
 - Network mounted file systems (NFS, SAMBA, etc)
 - Temporary partitions or logical volumes

```
$ sudo mount c8:/ISO_IMAGES /mnt
```

This directory is provided so that the system administrator can temporarily mount a filesystem when needed. A common use is for **network** filesystems, including:

- NFS
- Samba
- CIFS
- AFS

Historically, `/mnt` was also used for the kinds of files which are now mounted under `media` (or `/run/media`) in modern systems. Generally speaking, this directory should not be used by installation programs. Another temporary directory not currently being used serves better.

For the exclusive use of LFS301 corp class taught 06 to 09 December

2.13 /opt

/opt

- Used for the installation of add-on application software packages
- Static files for a package must be in `/opt/[somepackage]` directory

```
$ ls -l /opt
total 20
drwxr-xr-x  4 root root  4096 Feb  4  2020 brother
drwxr-xr-x  4 root root  4096 Sep  1  2019 google
drwxr-xr-x 29 root root 12288 Jan 19 17:06 zoom
```

This directory is designed for software packages that wish to keep all or most of their files in one isolated place rather than scatter them all over the system in directories shared by other software.

For example, if **dolphy_app** were the name of a package which resided under `/opt`, then all of its files should reside in directories under `/opt/dolphy_app`, including `/opt/dolphy_app/bin` for binaries and `/opt/dolphy_app/man` for any **man** pages.

This can make both installing and uninstalling software relatively easy, as everything is in one convenient isolated location in a predictable and structured manner. It also makes it easier for system administrators to determine the nature of each file within a package.

Note, however, if one uses packaging systems such as **RPM** and **APT**, as we shall discuss later, it is also easy to install and uninstall with a clear sense of file manifests and locations, without exhibiting such antisocial behavior.

In **Linux**, the `/opt` directory is often used by application providers with either proprietary software, or those who like to avoid complications of distribution variance. For example, on one system the packages are in `/opt/brother`, `/opt/zoom` and `/opt/google` and the latter has subdirectories for `chrome` and `earth`.

The directories `/opt/bin`, `/opt/doc`, `/opt/include`, `/opt/info`, `/opt/lib`, and `/opt/man` are reserved for local system administrator use. Packages may provide files which are linked or copied to these reserved directories, but the packages must also be able to function without the programs being in these special directories. Most systems do not populate these directories.

2.14 /proc

/proc

- A pseudo-file system: not stored anywhere on disk
- An interface to the kernel data structures
- Each active process has a subdirectory in this directory

This directory is the mount point for a **pseudo-filesystem** where all information resides only in memory, not on disk. Like `/dev` the `/proc` directory is empty on a non-running system.

The kernel exposes some important data structures through `/proc` entries. Additionally, each active process on the system has its own subdirectory that gives detailed information about the state of the process, the resources it is using and its history.

The entries in `/proc` are often termed **virtual files** and have interesting qualities. While most are listed as zero bytes in size, when viewed they can contain a large amount of information.

In addition, most of the time and date settings on virtual files reflect the current time and date, indicative of the fact they are constantly changing. In fact, the information in these files is obtained only when they are viewed, they are not being constantly or periodically updated.

Important pseudo-files including `/proc/interrupts`, `/proc/meminfo`, `/proc/mounts`, and `/proc/partitions`, provide an up-to-the-moment glimpse of the system's hardware.

Others, like `/proc/filesystems` and the `/proc/sys/` directory, provide system configuration information and interfaces.

For organizational purposes, files containing information on a similar topic are grouped into virtual directories and sub-directories. For instance, `/proc/scsi/` contains information for all physical **SCSI** devices. Likewise, the **process** directories contain information about each running process on the system.

We will extensively examine entries in `/proc` throughout this course, and we will take a more detailed look at them in upcoming chapters about kernel configuration and system monitoring.

For the exclusive use of LFS301 corp class taught 06 to 09 December

/proc Example

```
student@ubuntu:~$ ls -F /proc
1/   13/   200/  2288/  250/   33/   40/   5383/  99/      misc
10/   137/   201/  229/   251/   3315/  4047/  54/   990/      modules
100/   14/   202/   23/   2529/  3336/  41/   5410/  acpi/
1005/  1457/   203/  230/   2572/  3353/  4156/  5436/  buddyinfo
1007/  1465/   204/  231/   26/   3366/  4169/  5438/  bus/
1008/  1478/   205/  232/   27/   3394/  418/   581/   cgroups
1009/  15/   206/  233/   274/   34/   419/   583/   cmdline
101/   16/   207/  234/   2756/  3419/  42/   6/   consoles
1010/  1649/   208/  235/   276/   3430/  420/   601/   cpufreq
1011/  1780/   209/  236/   28/   3439/  422/   7/   crypto
1012/  1782/   21/   2368/  280/   3441/  43/   715/   devices
1013/  1798/   210/  237/   281/   35/   44/   717/   diskstats
1014/  18/   211/   2373/  2979/  3512/  45/   718/   dma
102/   1822/  212/   238/   298/   3513/  4595/  721/   driver/
1028/  188/   213/   2385/  2989/  3514/  4596/  723/   execdomains
103/   189/   214/   239/   30/   3515/  4599/  728/   fb
104/   19/   2142/  24/   300/   3532/  46/   731/   filesystems
105/   190/   215/   240/   3081/  3534/  4601/  733/   fs/
106/   191/   216/  241/   31/   3569/  47/   735/   interrupts
107/   1916/  217/  242/   32/   3581/  473/  743/   iomen
108/   192/  218/  243/   3228/  3589/  478/  746/   ioports
1083/  1922/  219/  2436/  3243/  36/   4792/  750/   irq/
1085/  193/  2196/  244/  3244/  3600/  483/  753/   kallsyms
109/   1930/  2197/  2443/  3245/  3613/  485/  755/   kcore
11/   194/   22/   2449/  3246/  3621/  486/  781/   keys
110/   1940/  220/  245/   3247/  3655/  492/  798/   key-users
1121/  195/   221/  2456/  3251/  37/   50/   8/   knsg
1134/  196/   222/  246/   3255/  370/   502/  828/   kpagegroup
1140/  197/   223/  2461/  3257/  38/   51/   9/   kpagecount
1141/  198/   224/  247/   3264/  39/   52/   96/   kpageflags
1147/  1986/  225/  248/   3268/  3910/  53/   98/   loadavg
116/   199/   226/  2488/  3272/  393/  5310/  981/   locks
1180/  2/   227/  249/   3274/  399/  5332/  983/   mdstat
12/   20/   228/  25/   3294/  4/   5358/  988/   meminfo
student@ubuntu:~$
```

Figure 2.6: /proc Directory

Here are the contents of the `/proc` directory belonging to one particular process.

```
student@ubuntu:~$ ls -F /proc/3589
attr/  coredump_filter/  cwd@/  environ/  limits/  loginuid/  map_files/  ns/  projid_map/  root@/  sched/  sessionid/  sessionstats/  syscalls/  task/  umountinfo/  oom_score/  oom_score_adj/  sessionid/  syscall
autogroup/  cpuset/  io/  mounts/  oom_score/  oom_score_adj/  sessionid/  syscall
auxv/  cwd@/  limits/  mountstats/  pagemap/  setgroups/  task/
cgroup/  environ/  loginuid/  net/  personality/  smaps/  timers
clear_refs/  exe@/  map_files/  ns/  projid_map/  stack/  timerslack_ns
cmdline/  fd/  maps/  numa_maps/  root@/  stat/  uid_map
comm/  fdfinfo/  mem/  oom_adj/  sched/  statm/  wchan
student@ubuntu:~$
```

Figure 2.7: /proc/[pid] Directory

One of many files showing important system information, `/proc/interrupts`.

```
x7:/home/coop>cat /proc/interrupts
File Edit View Search Terminal Help
CPU0    CPU1    CPU2    CPU3
0:      88      0       0       0   IR-IO-APIC  2-edge    timer
1:     566      1     2153      0   IR-IO-APIC  1-edge    i8042
8:      1       0       0       0   IR-IO-APIC  8-edge    rtc0
9:  17990     11     552      27  IR-IO-APIC  9-fasteoi  acpi
12:  64336     21   186157     20  IR-IO-APIC  12-edge   i8042
16:      0       0       0       0   IR-IO-APIC  16-fasteoi i801_smbus
120:     0       0       0       0   DMAR-MSI  0-edge    dmar0
121:     0       0       0       0   DMAR-MSI  1-edge    dmar1
122: 217295    1607    4039    59571 IR-PCI-MSI 376932-edge ahci[0000:00:17.0]
123:      3       0      46      0   IR-PCI-MSI 514048-edge snd_hda_intel:card0
124:  95360     74    49535    192 IR-PCI-MSI 327680-edge xhci_hcd
128: 591286     3    272983     2   IR-PCI-MSI 32768-edge i915
126:      84      16     149      20  IR-PCI-MSI 520192-edge emop0s3lf6
127: 225390     29     383      46  IR-PCI-MSI 2097152-edge iwlwifi
NMI:     24     120     130     119 Non-maskable interrupts
LOC: 2255506   2201465   2360863   2239138 Local timer interrupts
SPU:      0       0       0       0 Spurious interrupts
PMI:      24     120     130     119 Performance monitoring interrupts
IWI:      0       0       3       0 IRQ work interrupts
RTR:      24      3       0       0 APIC ICR read retries
RES: 185530 146421    95420    45924 Rescheduling interrupts
CAL:  76456   74989   78143    76663 Function call interrupts
TLB:  75401   73603   76833    75025 TLB shootdowns
ERR:      0       0       0       0
MIS:      0       0       0       0 Posted-interrupt notification event
PIN:      0       0       0       0 Posted-interrupt wakeup event
x7:/home/coop|
```

Figure 2.8: Contents of /proc/interrupts

For the exclusive use of LFS301 corp class taught 06 to 09 December

2.15 /sys

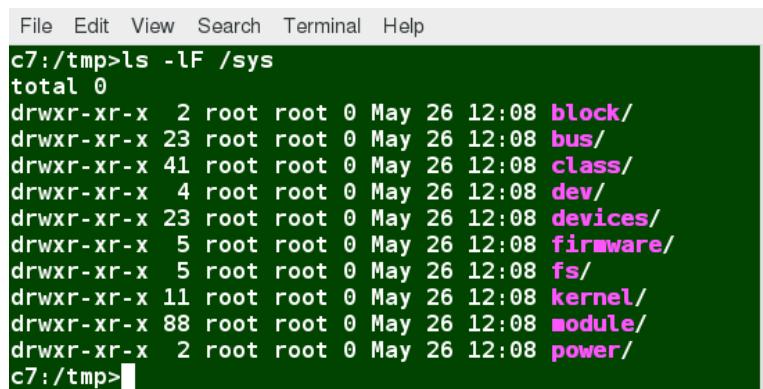
/sys

- Another pseudo-file system: not stored anywhere on disk
- Contains information about devices and drivers, kernel modules, system configuration structures
- Can be used to examine or modify system behavior and parameters
- Better controlled than [/proc](#)

This directory is the mount point for the **sysfs pseudo-filesystem** where all information resides only in memory, not on disk. Like [/dev](#) and [/proc](#), the [/sys](#) directory is empty on a non-running system.

sysfs is used to both gather information about the system, and modify its behavior while running. In that sense it resembles [/proc](#), but it is younger than and has adhered to strict standards about what kind of entries it can contain. For example, almost all pseudo-files in [/sys](#) contain only one line, or value; there are none of the long entries you can find in [/proc](#).

As with [/proc](#), we will examine entries in [/sys](#) throughout this course, and it will become relevant in upcoming chapters about kernel configuration and system monitoring.



```
File Edit View Search Terminal Help
c7:/tmp>ls -lF /sys
total 0
drwxr-xr-x 2 root root 0 May 26 12:08 block/
drwxr-xr-x 23 root root 0 May 26 12:08 bus/
drwxr-xr-x 41 root root 0 May 26 12:08 class/
drwxr-xr-x 4 root root 0 May 26 12:08 dev/
drwxr-xr-x 23 root root 0 May 26 12:08 devices/
drwxr-xr-x 5 root root 0 May 26 12:08 firmware/
drwxr-xr-x 5 root root 0 May 26 12:08 fs/
drwxr-xr-x 11 root root 0 May 26 12:08 kernel/
drwxr-xr-x 88 root root 0 May 26 12:08 module/
drwxr-xr-x 2 root root 0 May 26 12:08 power/
c7:/tmp>
```

Figure 2.9: /sys Directory

For the exclusive use of LFS301 corp class taught 06 to 09 December

2.16 /root

/root

- Used as the home directory for the root user

```
File Edit View Search Terminal Help
c7:/tmp>su
Password:
c7:/tmp>ls -saF /root
total 220
4 ./          4 .dbus/          4 .lesshst        4 shit
4 ../         4 Desktop/        4 .local/         4 .ssh/
4 anaconda-ks.cfg 4 Documents/   4 lttng-traces/  4 .tcshrc
24 .bash_history  4 Downloads/    4 .macromedia/   4 Templates/
4 .bash_logout    8 .emacs        4 Music/          4 Videos/
4 .bash_profile   4 .emacs.d/     4 perl5/         4 .vmware/
8 .bashrc         4 .esd auth    4 Pictures/      4 .xauth2xyWfQ
8 bashrc_ORIG    4 .galileo/     8 .pki/          4 .xautha52ibW
4 .basrc_ORIG    4 .gitk         4 Public/        4 .Xauthority
4 .cache/        8 .gnome2/      4 .rnd           4 .rpmbuild/
4 .config/       4 .ICEAuthority 4 sane/          4 .sanerun/
4 .cshrc         4 initial-setup-ks.cfg 4 .rpmbuild/
c7:/tmp>
```

Figure 2.10: /root Directory

This directory (pronounced “slash-root”) is the home directory for the root user.

The **root** account that owns this directory should only be used for actions which require superuser privilege. For those actions which can be done as a non-privileged user, use another account.

2.17 /sbin

/sbin

- Used for essential system and maintenance utilities
- Programs in this directory generally used by privileged users
- Executable files in this directory used to boot and mount `/usr` (in addition to those in `/bin`)
- Executable files in this directory used to perform system recovery operations (in addition to those in `/bin`)

/sbin and /usr/sbin

As mentioned earlier, some recent distributions have abandoned the strategy of separating `/sbin` and `/usr/sbin` (as well as `/bin` and `/usr/bin`) and just having one directory with symbolic links preserving a two directory view.

This directory contains binaries essential for booting, restoring, recovering, and/or repairing in addition to those binaries in the `/bin` directory. They also must be able to mount other filesystems on `/usr`, `/home` and other locations if needed, once the root filesystem is known to be in good health during boot.

The following programs should be included in this directory (if their subsystems are installed):

fdisk, fsck, getty, halt, ifconfig, init, mkfs, mkswap, reboot, route, swapon, swapoff, update.

```
$ ls -l /sbin
```

```
1 rwxrwxrwx 1 root root 8 Apr 23 2020 /sbin -> usr/sbin
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

/sbin Example

```
student@ubuntu:~$ ls /sbin
acpt_available    fscck.minix    lvextend    ntfsundelete    stop
agetty           fscck.msdos    lvm         on_ac_power    sulogin
also             fscck.nfs      lvmchange   osd_login      swaplabel
apm_available    fscck.vfat    lvmconf     pam_extrousers chkpwd
apparmor_parser  fscck.xfs     lvmconfig   pam_extrousers_update
badblocks        fsfreeze      lvdiskscan pam_extrousers_update
blkdiscard       fstab-decode  lvndump     pam_tally
blkid            fstrim       lvmetad     parted
blockdev         gdisk        lvmcold    partprobe
brctl            getcap       lvnsadc    pccardctl
bridge           getpcaps    lvreduce   pivot_root
brltty           getty        lvrenove   plipconfig
brltty-setup     halt        lvrename   poweroff
capsh            hdparm      lvresize  pvchange
cfdisk           ifconfig    lvs         pvck
cgdisk           ifdown      lvscan     pvcreate
cgmanager        ifquery     MAKEDEV   pvdisplay
cgproxy          ifup        mdadm     pvmove
chcpu            init       mdmon     pvrename
coldreboot       initctl    mil-tool  pvs
cryptdisks_start insmod     mkdosfs  pvs
cryptdisks_stop installkernel mke2fs  pvscan
cryptsetup       ip          mkfs     quotacheck
cryptsetup-reencrypt ipTables   mkfs.bfs  quotaoff
ctrlaltdel      ipTables-restore mkfs.cramfs  quotaoan
debugfs          ipTables-save mkfs.ext2  rarp
depmod           ipTables    mkfs.ext3  raw
dhclient         ipTables   mkfs.ext4  reboot
dhclient-script iptables-restore mkfs.ext4dev regdbdump
dneventd         iptables-save mkfs.fat  reload
dnsetup          iptunnel    mkfs.minix request-key
dosfsck          isosize     mkfs.msdos  resize2fs
dosfslabel       iw          mkfs.ntfs  resolvconf
dumpe2fs         iwconfig   mkfs.vfat  restart
e2fsck           iwevent    mkfs.xfs   rmmod
                                     ntfsundelete    stop
                                     on_ac_power    sulogin
                                     osd_login      swaplabel
                                     pam_extrousers chkpwd
                                     pam_extrousers_update
                                     pam_tally
                                     parted
                                     partprobe
                                     pccardctl
                                     pivot_root
                                     plipconfig
                                     poweroff
                                     pvchange
                                     pvck
                                     unix_chkpwd
                                     unix_update
                                     upstart
                                     upstart-dbus-bridge
                                     upstart-event-bridge
                                     upstart-file-bridge
                                     upstart-local-bridge
                                     upstart-socket-bridge
                                     upstart-udev-bridge
                                     ureadahead
                                     upstart
                                     vgcfgbackup
                                     vgcfgrestore
                                     vgchange
                                     vgck
                                     vgconvert
                                     vgcreate
                                     vgdisplay
                                     vgexport
                                     vgextend
```

To release input, press Ctrl+Alt

Figure 2.11: /sbin Directory: Ubuntu 18.04

Recent distribution versions of **RHEL**, **CentOS**, **Fedora**, and **Ubuntu** have symbolically linked `/sbin` and `/usr/sbin` so they are actually the same.

The above screenshot is from **Ubuntu 18.04**; later versions no longer have a separate un-linked directory.

2.18 /srv

/srv

- Site-specific variable data
- Data, scripts, for servers (http, ftp, etc.)
- Repositories for version control systems
- Packages should not place files there on installation

According to the **FHS**:

/srv contains site-specific data which is served by this system.

This main purpose of specifying this is so that users may find the location of the data files for particular service, and so that services which require a single tree for readonly data, writable data and scripts (such as cgi scripts) can be reasonably placed.

The methodology used to name subdirectories of /srv is unspecified as there is currently no consensus on how this should be done. One method for structuring data under /srv is by protocol, e.g.. ftp, rsync, www, and cvs.

Some system administrators (and distributions) swear by the use of [/srv](#); others ignore it. There is often confusion about what is best to go in [/var](#) as opposed to [/srv](#).

For the exclusive use of LFS301 corp class taught 06 to 09 December

2.19 /tmp

/tmp

- Used to store temporary files
- Assume that any files stored here are subject to deletion
 - **Ubuntu** wipes `/tmp` on every system reboot!
- Beware of systems (such as **Fedora**) that mount `/tmp` as a pseudo-filesystem in memory.
 - Convert to a real filesystem on storage with:
`$ sudo systemctl mask tmp.mount`
and a reboot

Used to store temporary files, and can be accessed by any user or application. However, any files on `/tmp` can not be depended on to stay around for a long time:

- Some distributions run automated **cron** jobs which remove any files older than 10 days typically, unless the purge scripts have been modified to exclude them.
- Some distributions remove the contents of `/tmp` with every reboot. This has been the **Ubuntu** policy.
- Some modern distributions utilize a virtual filesystem using the `/tmp` directory only as a mount point for a **ram disk** using the **tmpfs** filesystem. This is the default policy on **Fedora** systems. When the system reboots, all information is thereby lost; `/tmp` is indeed temporary!

In the last case, one must avoid creating large files on `/tmp`; they will actually occupy space in memory rather than disk, and it is easy to harm or crash the system through memory exhaustion. While the guideline is for applications to avoid putting large files in `/tmp`, there are plenty of applications that violate this policy and which make large temporary files in `/tmp`; even if it is possible to put them somewhere else (perhaps by specifying an environment variable), many users are not aware of how to configure this and all users have access to `/tmp`.

This policy can be canceled on systems using **systemd** such as **Fedora**, by issuing the command:

```
$ sudo systemctl mask tmp.mount
```

followed by a system reboot.

For the exclusive use of LFS301 corp class taught 06 to 09 December

2.20 /usr

/usr

Table 2.3: Directories Under /usr

<code>/usr/bin</code>	Non-essential command binaries
<code>/usr/etc</code>	Non-essential configuration files (usually empty)
<code>/usr/games</code>	Game data
<code>/usr/include</code>	Header files used to compile applications.
<code>/usr/lib</code>	Library files
<code>/usr/lib64</code>	Library files for 64-bit
<code>/usr/local</code>	Third-level hierarchy (for machine local files)
<code>/usr/sbin</code>	Non-essential system binaries
<code>/usr/share</code>	Read-only architecture independent files
<code>/usr/src</code>	Source code and headers for the Linux kernel
<code>/usr/tmp</code>	Secondary temporary directory

The `/usr` directory can be thought of as a **secondary hierarchy**; it is used for files which are not needed for system booting. Indeed, `/usr` need not reside in the same partition as the root directory, and can be shared among hosts using the same system architecture across a network.

Software packages should not create subdirectories directly under `/usr`. Some symbolic links may exist to other locations for compatibility purposes.

This directory is typically read-only data. It contains binaries which are not needed in single user mode. It contains the `/usr/local` directory where local binaries and such may be stored. **man** pages are stored under `/usr/share/man`.



Very Important

Some recent distributions have abandoned the strategy of separating `/bin`, `/sbin`, `/lib`, and `/lib64` from their partners under `/usr`, using symbolic links to join them. They view the time-honored concept of enabling the possibility of placing `/usr` on a separate partition to be mounted after boot as obsolete.

For the exclusive use of LFS301 corp class taught 06 to 09 December

2.21 /var

/var

- Contains variable data files: Cannot be mounted read-only
 - Log files
 - Spool directories and files
 - Administrative data files
 - Transient and temporary files, such as cache contents

Table 2.4: Directories Under /var

Directory	Purpose
<code>/var/ftp</code>	Used for ftp server base.
<code>/var/lib</code>	Persistent data modified by programs as they run.
<code>/var/lock</code>	Lock files used to control simultaneous access to resources.
<code>/var/log</code>	Log files.
<code>/var/mail</code>	User mailboxes.
<code>/var/run</code>	Information about the running system since the last boot.
<code>/var/spool</code>	Tasks spoiled or waiting to be processed, such as print queues.
<code>/var/tmp</code>	Temporary files to be preserved across system reboot. Sometimes linked to <code>/tmp</code> .
<code>/var/www</code>	Root for website hierarchies.

This directory contains **variable** (or **volatile**) data files that change frequently during system operation. Obviously `/var` cannot be mounted as a read-only filesystem. For security reasons, it is often considered a good idea to mount `/var` as a separate filesystem. Furthermore, if the directory gets filled up it should not lock up the system.

`/var/log` is where most of the log files are located. `/var/spool` is where local files for processes such as mail, printing, and cron jobs are stored while awaiting action.

```
student@debian:~$ ls -lF /var
total 48
drwxr-xr-x 2 root root 4096 May 26 12:23 backups/
drwxr-xr-x 14 root root 4096 Jan 16 2016 cache/
drwxr-xr-x 2 root root 4096 Nov 24 2015 crash/
drwxr-xr-x 2 root root 4096 Apr 26 2015 games/
drwxr-xr-x 62 root root 4096 May 1 11:21 lib/
drwxrwsr-x 2 root staff 4096 Nov 30 2014 local/
lrwxrwxrwx 1 root root 9 Apr 26 2015 lock -> /run/lock/
drwxr-xr-x 14 root root 4096 May 26 12:24 log/
drwxrwsr-x 2 root mail 4096 May 26 12:24 mail/
drwxr-xr-x 2 root root 4096 Apr 26 2015 opt/
lrwxrwxrwx 1 root root 4 Apr 26 2015 run -> /run/
drwxr-xr-x 7 root root 4096 Oct 17 2016 spool/
drwxrwxrwt 14 root root 4096 May 26 12:19 tmp/
drwxr-xr-x 3 root root 4096 Nov 2 2015 www/
student@debian:~$
```

Figure 2.12: /var Directory

For the exclusive use of LFS301 corp class taught 06 to 09 December

2.22 /run

/run

- Pseudo-filesystem, exists only in memory
- Used for transient run-time files
- Often used to mount removable media

The purpose of `/run` is to store **transient** files; those that contain run-time information, which may need to be written early in system startup, and which do not need to be preserved when rebooting.

Generally `/run` is implemented as an empty mount point, with a `tmpfs` ram disk (like `/dev/shm`) mounted there at runtime. Thus this is a pseudo-filesystem existing only in memory.

Some existing locations, such as `/var/run` and `/var/lock`, will be now just symbolic links to directories under `/run`. Other locations, depending on distribution taste, may also just point to locations under `/run`.

```
student@debian:~$ ls -rF /run
vsftpd/          screen/          lvm/
utmp             rsyslogd.pid    log/
user/            rpc.statd.pid   Lock/
udisks2/         rpc_pipefs/    libvirtd.pid
udev/            rpcbind.sock=  libvirt/
tmpfiles.d/      rpcbind.pid    initramfs/
systemd/         rpcbind.lock   initctl@
sshda.pid        rpcbind/       gdm3.pid
sshd/            NetworkManager/ gdm3/
sm-notify.pid    network/      dovecot/
shm@             mount/        dmeventd-server|
setrans/          mlocate.daily.lock dmeventd-client|
sendsig.omit.d/   adm/         dhclient-eth0.pid
student@debian:~$
```

Figure 2.13: /run Directory

For the exclusive use of LFS301 corp class taught 06 to 09 December

2.23 Labs



Video Demonstration Resources

[using_linux_distros_demo.mp4](#)
[using_disk_usage_demo.mp4](#)

Exercise 2.1: Sizes of the Default Linux Directories

Use the **du** utility to calculate the overall size of each of your system's top-level directories.

Type the command:

```
$ du --help
```

for hints on how to obtain and display this result efficiently.

Solution 2.1

To obtain a full list of directories under / along with their size:

```
$ sudo du --max-depth=1 -hx /
```

1	4.3M	/home
2	16K	/lost+found
3	39M	/etc
4	4.0K	/srv
5	3.6M	/root
6	178M	/opt
7	138M	/boot
8	6.1G	/usr
9	1.1G	/var
10	16K	/mnt
11	4.0K	/media
12	869M	/tmp
13	8.4G	/

Where we have used the options:

- **--maxdepth=1**: Just go down one level from / and sum up everything recursively underneath in the tree.
- **-h**: Give human-readable numbers (KB, MB, GB).
- **-x** Stay on one filesystem; don't look at directories that are not on the / partition. In this case that means ignore:

```
/dev /proc /run /sys
```

because these are pseudo-filesystems which exist in memory only; they are just empty mount points when the system is not running. Because this was done on a **RHEL** system, the following mount points are also not followed:

```
/bin /sbin /lib /lib64
```

since they are just symbolically linked to their counterparts under /usr.

Exercise 2.2: Touring the /proc Filesystem



Please Note

Exactly what you see in this exercise will depend on your kernel version, so you may not match the output shown precisely.

For the exclusive use of LFS301 corp class taught 06 to 09 December

- As root, **cd** into `/proc` and do a directory listing. This should display a number of files and directories:

```
$ cd /proc
$ ls -F
```

1	1/	128/	1510/	20/	2411/	30895/	53/	6925/	802/	951/	kmsg
2	10/	129/	1511/	2015/	2425/	31/	54/	7/	81/	952/	kpagecgroup
3	1002/	13/	1512/	2022/	2436/	31449/	55/	70/	813/	957/	kpagecount
4	1007/	130/	1513/	2023/	2444/	32/	56/	702/	814/	97/	kpageflags
5	10540/	131/	1514/	20300/	2451/	33/	58/	709/	816/	9742/	loadavg
6	10590/	13172/	152/	20354/	2457/	34/	585/	71/	817/	98/	locks
7	10798/	132/	15552/	20380/	2489/	35/	59/	718/	82/	99/	meminfo
8	10805/	133/	15663/	20388/	25/	36/	60/	719/	83/	9923/	misc
9	10806/	134/	15737/	20392/	2503/	37/	61/	72/	834/	acpi/	modules
10	10809/	135/	159/	20396/	2504/	374/	6193/	721/	835/	asound/	mounts@
11	10810/	136/	15981/	2086/	2531/	379/	62/	723/	84/	buddyinfo	mtrr
12	10813/	137/	16/	2090/	2546/	38/	63/	725/	841/	bus/	net@
13	10894/	138/	162/	211/	2549/	380/	634/	727/	842/	cgroups	pagetypeinfo
14	10925/	1384/	1632/	22/	2562/	40/	64/	73/	85/	cmdline	partitions
15	10932/	1385/	1636/	2205/	25794/	41/	65/	7300/	857/	config.gz	sched_debug
16	10934/	1387/	166/	2209/	26/	42/	662/	74/	86/	consoles	scsi/
17	10935/	139/	1670/	2212/	2610/	43/	663/	757/	864/	cpuinfo	self@
18	10941/	1390/	17/	2232/	26108/	44/	665/	758/	867/	crypto	slabinfo
19	10983/	1393/	17271/	2238/	2619/	4435/	666/	76/	87/	devices	softirqs
20	10998/	14/	17361/	2296/	2624/	45/	67/	761/	88/	diskstats	stat
21	11/	140/	1793/	2298/	2627/	46/	670/	762/	881/	dma	swaps
22	11047/	1410/	18/	23/	2644/	468/	671/	765/	886/	driver/	sys/
23	1105/	1415/	1831/	23042/	2645/	47/	673/	766/	887/	execdomains	sysrq-trigger
24	1121/	1429/	18880/	2344/	2679/	470/	674/	768/	888/	fb	sysvipc/
25	1123/	1437/	18903/	2348/	27/	484/	678/	769/	889/	filesystems	thread-self@
26	1135/	1445/	19/	2353/	2706/	49/	679/	77/	89/	fs/	timer_list
27	11420/	146/	19392/	2354/	2762/	492/	68/	771/	9/	interrupts	timer_stats
28	11499/	1463/	19488/	2365/	28/	493/	682/	78/	90/	iomem	tty/
29	11515/	147/	1954/	23683/	2858/	5/	683/	79/	92/	iports	uptime
30	11530/	1476/	1963/	2370/	28730/	50/	686/	793/	921/	irq/	version
31	1163/	148/	19727/	2372/	28734/	51/	687/	794/	928/	kallsyms	vmallocinfo
32	1164/	1485/	19734/	2374/	29/	510/	69/	8/	930/	kcore	vmstat
33	12/	149/	19984/	24/	2973/	514/	690/	80/	931/	keys	zoneinfo
34	127/	15/	2/	2406/	3/	52/	691/	801/	944/	key-users	

Notice many of the directory names are numbers; each corresponds to a running process and the name is the **process ID**. An important subdirectory we will discuss later is `/proc/sys`, under which many system parameters can be examined or modified.

- View the following files:

- `/proc/cpuinfo`:
- `/proc/meminfo`:
- `/proc/mounts`:
- `/proc/swaps`:
- `/proc/version`:
- `/proc/partitions`:
- `/proc/interrupts`:

The names give a pretty good idea about what information they reveal.

Note that this information is not being constantly updated; it is obtained only when one wants to look at it.

- Take a peek at any random process directory (if it is not a process you own some of the information might be limited unless you use **sudo**):

```
$ ls -F 4435
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

```
1 attr/      coredump_filter  gid_map      mountinfo   oom_score_adj  sessionid  syscall
2 autogroup  cpuset          io           mounts      pagemap       setgroups   task/
3 auxv       cwd@            limits        mountstats  personality   smaps      timerslack_ns
4 cgroup     environ         loginuid     net/        projid_map    stack      uid_map
5 clear_refs exe@           map_files/   ns/         root@        stat       wchan
6 cmdline    fd/             maps          oom_adj    sched        statm
7 comm       fdinfo/        mem          oom_score  schedstat   status
```

Take a look at some of the fields in here such as: cmdline, cwd, environ, mem, and status

Chapter 3

Processes



3.1	Programs and Processes	56
3.2	Process Limits	60
3.3	Creating Processes	63
3.4	Process States	65
3.5	Execution Modes	66
3.6	Daemons	69
3.7	niceness	70
3.8	Libraries	72
3.9	Labs	75

3.1 Programs and Processes

What is a Program?

- A **program** is a set of instructions to be carried out (plus any data required to do so)
- May consist of machine level instructions run directly by a CPU
- May consist of a list of commands to be interpreted by another program
- Programmers use various languages (such as **C**, **C++**, **Perl**, and many more) to code instructions in a program

programs vs scripts

People often distinguish between programs, which are **compiled** into a binary executable form; and **scripts**, which need to be run by an interpreter such as **bash**, **Python** or **Perl**.

A program is a set of instructions, along with any internal data used while carrying the instructions out. Programs may also use external data.. Internal data might include text strings inside the program which are used to display user prompts. External data might include data from a database.

Many user commands such as **ls**, **cat**, and **rm** are programs which are external to the operating system kernel, or shell (in other words they have their own executable program on disk).

For the exclusive use of LFS301 corp class taught 06 to 09 December

What is a Process?

- A **process** is a program being executed
- **Linux** creates a new process for every program that is executed or run
- Several processes may be executing the same program at the same time
- Primary purpose of the operating system is to manage the execution of processes on behalf of users



Multithreading

A program may be composed of multiple simultaneous **threads**, each of which is considered as its own process.

A process is an instance of a program in execution. It may be in a number of different states, such as running or sleeping.

Every process has a **pid** (Process ID), a **ppid** (Parent Process ID), and a **pgid** (Process Group ID). In addition every process has program code, data, variables, file descriptors, and an environment.

Processes are controlled by **scheduling**, which is completely preemptive. Only the kernel has the right to preempt a process; they cannot do it to each other.

For historical reasons, the largest **pid** has been limited to a 16-bit number, or 32768. It is possible to alter this value by changing `/proc/sys/kernel/pid_max`, since it may be inadequate for larger servers. As processes are created eventually they will reach `pid_max`, at which point they will start again at `PID = 300`.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Attributes of a Process

- All processes have certain attributes:
 - The program being executed
 - Context (state)
 - Permissions
 - Associated resources

Every process is executing some program. At any given moment, the process may take a snapshot of itself by trapping the state of its CPU registers, where it is executing in the program, what is in the process' memory, and other information. This is the **context** of the process.

Since processes can be scheduled in and out when sharing CPU time with others (or have to be put to sleep while waiting for some condition to be fulfilled, such as the user to make a request or data to arrive), being able to store the entire context when swapping out the process and being able to restore it upon execution resumption is critical to the kernel's ability to do **context switching**.

Every process has permissions based on which user has called it to execute. It may also have permissions based on who owns its program file. Programs which are marked with an "s" execute bit have a different "effective" user id than their "real" user id. These programs are referred to as setuid programs. They run with the user-id of the user who owns the program, where a non-setuid program runs with the permissions of the user who starts it. Setuid programs owned by root can be a security problem.

The **passwd** program is an example of a setuid program. It is runnable by any user. When a user executes this program, the process runs with root permission in order to be able to update the write-restricted files, `/etc/passwd` and `/etc/shadow`, where the user passwords are maintained.

Note that every process has resources such as allocated memory, file handles, etc.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Process Resource Isolation

- Kernel isolates processes from each other (by default)
 - Promotes security
 - Increase stability
- Kernel does not allow processes to access arbitrary hardware resources
 - Access to hardware managed by kernel
 - Access through **system calls**

When a process is started it is isolated in its own user space to protect it from other processes. This promotes security and creates greater stability.

Processes do not have direct access to hardware. Hardware is managed by the kernel so a process must use **system calls** to indirectly access hardware. System calls are the fundamental interface between an application and the kernel.

For the exclusive use of LFS301 corp class taught 06 to 09 December

3.2 Process Limits

Controlling Processes with ulimit

- Built in **bash** command
- Displays or resets process resource limits
- Use to **restrict** capabilities so individual user and/or processes cannot exhaust system resources such as:
 - Memory
 - CPU time
 - Number of active processes or files
- System administrator can set system-wide limits for all users

A system administrator may need to change some of these values in either direction:

- To **restrict** capabilities so an individual user and/or process can not exhaust system resources, such as memory, cpu time or the maximum number of processes on the system.
- To **expand** capabilities so a process does not run into resource limits; for example, a server handling many clients may find the default of 1024 open files makes its work impossible to perform.

```
student@debian:~$ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority     (-e) 0
file size               (blocks, -f) unlimited
pending signals          (-i) 14530
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files              (-n) 65536
pipe size               (512 bytes, -p) 8
POSIX message queues    (bytes, -q) 819200
real-time priority      (-r) 0
stack size              (kbytes, -s) 8192
cpu time                (seconds, -t) unlimited
max user processes      (-u) 14530
virtual memory           (kbytes, -v) unlimited
file locks              (-x) unlimited
student@debian:~$
```

Figure 3.1: **ulimit**

For the exclusive use of LFS301 corp class taught 06 to 09 December

Setting Limits

- Set any particular limit by doing:

```
$ ulimit [options] [limit]
```

as in

```
$ ulimit -n 1600
```

which would increase the maximum number of file descriptors to 1600.

- Make persistent by modifying `/etc/security/limits.conf` and rebooting

Note that the changes only affect the current shell. To make changes that are effective for all logged-in users, one needs to modify `/etc/security/limits.conf`, a very nicely self-documented file, and then reboot.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Hard and Soft Limits

There are two kinds of limits:

- **Hard**: Maximum value, set only by the root user, that a user can raise the resource limit to.
- **Soft**: Current limiting value, which a user can modify but can not exceed the hard limit.

```
$ ulimit -H -n
```

```
1 4096
```

```
$ ulimit -S -n
```

```
1 1024
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

3.3 Creating Processes

Creating Processes

- First user process is called **init** and has pid = 1
- Subsequent processes are **forked** from **init** or other running processes
- Forking from a **parent** produces a **child** process, in every way a equal and a peer of the parent.
- If the parent dies, the child is “adopted” by **init**
- There are also kernel-created processes; these have names surrounded by [...] in the output from **ps**



Please Note

systemd-based systems run a special process named **kthreadd** with pid=2 whose job is to adopt orphaned children, who will then show ppid=2.

An average **Linux** system is always creating new processes. This is often called **forking**; the original parent process keeps running while the new child process starts.

Often rather than just a fork, one follows it with an **exec**, where the parent process terminates and the child process inherits the process ID of the parent. The term **fork and exec** is used so often people think of it sometimes as one word.

Older **UNIX** systems often used a program called **spawn** which is similar in many ways to fork and exec, but differs in details. It is not part of the **POSIX** standard and is not a normal part of **Linux**.

To see how new processes may start, consider a web server that handles many clients. It may launch a new process every time a new connection is made with a client. On the other hand, it may simply start only a new **thread** as part of the same process; in **Linux** there really is not much difference on a technical level between creating a full process or just a new thread as each mechanism takes about the same time and uses roughly the same amount of resources.

As another example, the **sshd** daemon is started when the **init** process executes the **sshd** init script, which then is responsible for launching the **sshd daemon**. This daemon process listens for **ssh** requests from remote users.

When a request is received, **sshd** creates a new copy of itself to service the request. Each remote user gets their own copy of the **sshd** daemon running to service their remote login. The **sshd** process will start the login program to validate the remote user. If the authentication succeeds, the login process will fork off a shell (say **bash**) to interpret the user commands, and so on.

Internal kernel processes take care of maintenance work, such as making sure buffers get flushed out to disk, that the load on different CPUs is balanced evenly, that device drivers handle work that has been queued up for them to do, etc. These processes often run as long as the system is running, sleeping except when they have something to do.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Creating Processes in a Command Shell

When a user executes a command in a command shell interpreter, such as **bash**:

- A new process is created (forked from the user's login shell)
- A wait system call puts the parent shell process to sleep
- The command is loaded onto the child process's space via the **exec** system call, replacing **bash**
- The command completes executing, and the child process dies via the exit system call
- The parent shell is re-awakened by the death of the child process and proceeds to issue a new shell prompt
- The parent shell then waits for the next command request from the user, at which time the cycle will be repeated

If a command is issued for **background** processing (by adding an ampersand (&) at the end of the command line) the parent shell skips the wait request and is free to issue a new shell prompt immediately allowing the background process to execute in parallel. Otherwise, for **foreground** requests, the shell waits until the child process has completed or is stopped via a signal.

Some shell commands (such as **echo** and **kill**) are built into the shell itself and do not involve loading of program files. For these commands, neither a **fork** nor an **exec** is issued for the execution.

3.4 Process States

Process States

- Processes are in one of several possible states
 - **Running**: Executing on a CPU, or ready to execute when O/S permits
 - **Waiting**: Waiting for information from disk or some external source; also called **Sleeping**
 - **Stopped**: May be stopped by debugger or Ctrl-Z
 - **Zombie**: Gone, but not properly removed by parent process

A process in the **Running** state is either currently receiving attention from the CPU (in other words, executing) or waiting in a queue ready to run. A queued process will be selected to run when its priority has elevated it to first in the queue and the operating system has an idle CPU.

A process in the **Waiting** (or **Sleeping**) state is waiting on a request that it has made (usually I/O) and cannot proceed further until the request is completed. When the request is completed, an event interrupts the O/S to allow a CPU to be reassigned to the waiting process and it will continue processing.

A process may be **Stopped**, meaning execution of instructions has been suspended. This state is commonly experienced when a programmer wants to examine the executing programs memory, CPU registers, flags or other attributes. Once this is done the process may be resumed.

A **Zombie** state is entered when a process terminates. Zombie processes are important, because each Zombie continues to take up space in the O/S's process table. The process table has an entry for each active process in the system. A Zombie process has all of its resources released except its process table entry.

The scheduler manages all of the processes. Process state is reported by process listings.

For the exclusive use of LFS301 corp class taught 06 to 09 December

3.5 Execution Modes

Two Execution Modes

- User Mode
 - Running with restricted privileges
- System (or Kernel) Mode
 - Running with full unrestricted privileges

At any given time a process (or any particular thread of a multi-threaded process) may be executing in either **user mode** or **system mode**, which is usually called **kernel mode** by kernel developers.

What instructions can be executed depends on the mode and is enforced at the hardware, not software, level.

The mode is not a state of the system; it is a state of the processor as in a multi-core or multi-CPU system each unit can be in its own individual state.

In Intel parlance, user mode is also termed **Ring 3** and system mode is termed **Ring 0**.

For the exclusive use of LFS301 corp class taught 06 to 09 December

User Mode

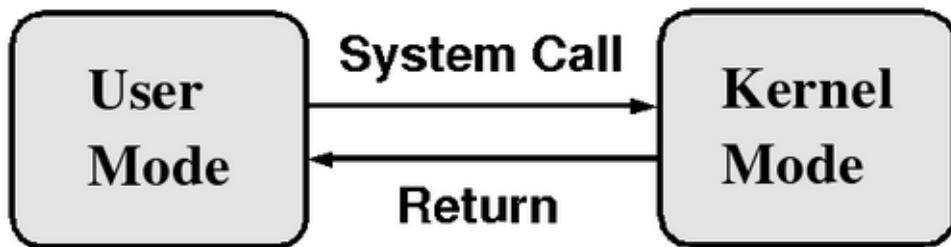


Figure 3.2: User and Kernel Modes and System Calls

- **User mode** has fewer privileges than **kernel mode**
- Regular program code executes in this mode

Except when executing a **system call** (described in the next section) processes execute in **user mode**, where they have lesser privileges than in kernel mode.

When a process is started it is isolated in its own user space to protect it from other processes. This promotes security and creates greater stability. This is sometimes called **process resource isolation**.

Each process executing in user mode has its own memory space, parts of which may be shared with other processes; except for the shared memory segments, a user process is not able to read or write into or from the memory space of any other process.

Even a process run by the root user or as a setuid program runs in user mode except when jumping into a system call, and has only limited ability to access hardware.

System (Kernel) Mode

- Entered when a program makes a system call
- Lasts until the system call exits
- More privileges than user mode
- Processes in system mode are running kernel code
 - Obtaining disk access (read, write)
 - Requesting additional memory(brk, sbrk)
 - Changing the current working directory (cd)

In system (kernel) mode the CPU has full access to all hardware on the system, including peripherals, memory, disks, etc. If an application needs access to these resources it must issue a **system call**, which causes a **context switch** from user mode to kernel mode. This procedure must be followed when reading and writing from files, creating a new process, etc.

Application code never runs in kernel mode, only the system call itself which is kernel code. When the system call is complete, a return value is produced and the process returns to user mode with the inverse context switch.

There are other times when the system is in kernel mode that have nothing to do with processes, such as when handling hardware interrupts or running the scheduling routines and other management tasks for the system.

For the exclusive use of LFS301 corp class taught 06 to 09 December

3.6 Daemons

Daemons

- A **daemon** is a background process whose sole purpose is to provide some specific service to users of the system.
- Can be quite efficient because they only operate when needed.
- Many are started at boot time.
- Names often (but not always) end with **d**: Examples include **httpd** and **systemd-udevd**.
- May respond to external events (**systemd-udevd**) or elapsed time (**crond**).
- Generally have no controlling terminal and no standard input/output devices.
- Sometimes provide better security control.

A daemon process usually is a background process that provides a specific service to users of the system. Its sole purpose on the system is to provide that specific service. Some examples include **xinetd**, **httpd**, **lpd**, and **vsftpd**.

3.7 niceness

Using nice to Set Priorities

- Every process has a **nice value** which affects the process's execution priority
- **nice** can raise or lower a process' priority by adjusting its nice value
- Higher nice values lower the priority
- Lower nice values raise the priority
- Non-privileged users can only raise the nice value
- Only the superuser can lower the nice value

Nice values range in value from -20 (lowest nice, highest priority) to 19 (highest nice, lowest priority). Note that only the superuser can lower a nice value but any user can raise their process' nice value. By default, a process has a nice value of 0 which is inherited from the shell.

nice can be used to run a process with a specific nice value:

```
$ nice -n 10 myprog
```

runs **myprog** with nice value of 10

```
$ nice -n 19 myprog
```

runs **myprog** with lowest priority, nice value of 19

```
$ sudo nice -n -20 myprog
```

runs **myprog** with highest priority, nice value of -20

If you do not give a nice value the default is to increase the niceness by 10. If you give no arguments at all, you report your current niceness.

Note that increasing the niceness of a process does not mean it will not run; it may even get all the CPU time if there is nothing else to compete with.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Modifying the Nice Value

- **renice** can change the nice value of a running process

```
$ renice --help
```

Usage:

```
renice [-n] <priority> [-p|--pid] <pid>...
renice [-n] <priority> -g|--pgrp <pgid>...
renice [-n] <priority> -u|--user <user>...
```

- Raise the nice value of pid 20003 to 5 :

```
$ renice +5 -p 20003
```

- Only the superuser can lower the nice value of processes

renice is used to raise or lower the nice value of an already running process. It basically lets you change the nice value on the fly.

By default only a superuser can decrease the niceness; i.e., increase the priority. However it is possible to give normal users the ability to decrease their niceness within a predetermined range, by editing [/etc/security/limits.conf](#).

After a non-privileged user has increased the nice value, only a superuser can lower it back.

To change the niceness of an already running process it is easy to use the **renice** command, as in:

```
$ renice +3 13848
```

which will reset the niceness to 3 for the process with **pid = 13848**. More than one process can be done at the same time and there are some other options, so see **man renice**.

For the exclusive use of LFS301 corp class taught 06 to 09 December

3.8 Libraries

Static and Shared Libraries

- **Static Libraries**

- Code compiled into programs
- Do not get updated library features without recompile

- **Shared Libraries**

- Code loaded at run time
- Updated library code is used automatically
- Shared libraries are more efficient
- Also called **DLLs (Dynamic Link Library)**.

```
c8:/usr/lib64>ls -l libpthread.*
```

```
1 -rwxr-xr-x 1 root root 320504 Jan  7 11:20 libpthread-2.28.so
2 -rw-r--r-- 1 root root 993146 Jan  7 11:21 libpthread.a
3 lrwxrwxrwx 1 root root      27 Jan  7 11:08 libpthread.so -> ../../lib64/libpthread.so.0
4 lrwxrwxrwx 1 root root      18 Jan  7 11:09 libpthread.so.0 -> libpthread-2.28.so
```

Programs are built using **libraries** of code, developed for multiple purposes and used and reused in many contexts.

There are two types of libraries:

- **Static**

The code for the library functions is inserted in the program at **compile time**, and does not change thereafter, even if the library is updated.

- **Shared**

The code for the library functions is loaded into the program at **run time**, and if the library is changed later, the running program runs with the new library modifications.

Using shared libraries is more efficient because they can be used by many applications at once; memory usage, executable sizes, and application load time are reduced.

Shared Libraries are also called **DLLs (Dynamic Link Library)**.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Shared Library Versions

- Careful versioning needed
- Avoid **DLL Hell**
- Programs can request specific **major** and/or **minor** library versions
- Some applications use only static libraries to avoid problems, but can lag in bug and security fixes

Shared libraries need to be carefully versioned. If there is a significant change to the library and a program is not equipped to handle it, serious problems can be expected. This is sometimes known as **DLL Hell**.

Therefore, programs can request a specific **major** library version rather than the latest one on the system. However, usually the program will use the latest **minor** version available.

Some application providers will use static libraries bundled into the program to avoid these problems. However, if there are improvements or bugs and security holes fixed in the libraries, they may not make it into the applications in timely fashion.

Shared libraries have the extension **.so**. Typically the full name is something like **libc.so.N** where **N** is a major version number.

Under **Linux** shared libraries are carefully versioned. For example:

```
c8:/usr/lib64>ls -l libcrypt.*  
1 -rw-r--r-- 1 root root 233788 May 10 2019 libcrypt.a  
2 lwxrwxrwx 1 root root 17 Aug 12 2018 libcrypt.so -> libcrypt.so.1.1.0  
3 lwxrwxrwx 1 root root 17 Aug 12 2018 libcrypt.so.1 -> libcrypt.so.1.1.0  
4 -rwxr-xr-x 1 root root 142712 Aug 12 2018 libcrypt.so.1.1.0
```

so a program that just asks for **libcrypt** gets **libcrypt.so** and the others for specific major and minor versions. Also note **libcrypt.a** is available for a program that wants to link to the static version of the library.

Finding Shared Libraries

- Shared libraries need to be found at run time
- **ldd** shows shared libraries used by program
- **ldconfig** uses `/etc/ld.so.conf` to cache where to look.
 - Run at boot or any time later
- Can specify `LD_LIBRARY_PATH` to override default directory search

ldd ascertains which shared libraries an executable requires, showing the **soname** of the library and what file it points to:

```
student@ubuntu: $ ldd /usr/bin/vi
1      linux-vdso.so.1 (0x00007ffd2ae93000)
2      libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f9c9275a000)
3      libtinfo.so.6 => /lib/x86_64-linux-gnu/libtinfo.so.6 (0x00007f9c9272a000)
4      libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007f9c926ff000)
5      libcanberra.so.0 => /lib/x86_64-linux-gnu/libcanberra.so.0 (0x00007f9c926ec000)
6      libacl.so.1 => /lib/x86_64-linux-gnu/libacl.so.1 (0x00007f9c926e1000)
7      libgpm.so.2 => /lib/x86_64-linux-gnu/libgpm.so.2 (0x00007f9c924db000)
8      libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f9c924d3000)
9      libpython3.8.so.1.0 => /lib/x86_64-linux-gnu/libpython3.8.so.1.0 (0x00007f9c91f7d000)
10     libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f9c91f5a000)
11     libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f9c91d68000)
12     /lib64/ld-linux-x86-64.so.2 (0x00007f9c92b98000)
13     libpcre2-8.so.0 => /lib/x86_64-linux-gnu/libpcre2-8.so.0 (0x00007f9c91cd8000)
14     libvorbisfile.so.3 => /lib/x86_64-linux-gnu/libvorbisfile.so.3 (0x00007f9c91ccd000)
15     libtdb.so.1 => /lib/x86_64-linux-gnu/libtdb.so.1 (0x00007f9c91cb1000)
16     libltdl.so.7 => /lib/x86_64-linux-gnu/libltdl.so.7 (0x00007f9c91ca6000)
17     libexpat.so.1 => /lib/x86_64-linux-gnu/libexpat.so.1 (0x00007f9c91c78000)
18     libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007f9c91c5c000)
19     libutil.so.1 => /lib/x86_64-linux-gnu/libutil.so.1 (0x00007f9c91c57000)
20     libvorbis.so.0 => /lib/x86_64-linux-gnu/libvorbis.so.0 (0x00007f9c91c27000)
21     libogg.so.0 => /lib/x86_64-linux-gnu/libogg.so.0 (0x00007f9c91c1a000)
```

ldconfig is generally run at boot time (but can be run anytime), and uses the `/etc/ld.so.conf`, which lists the directories that will be searched for shared libraries.

Besides searching the database built up by **ldconfig**, the linker will first search any directories specified in the environment variable `LD_LIBRARY_PATH`, a colon separated list of directories, as in the `PATH` variable. So you can do either of:

```
$ LD_LIBRARY_PATH=$HOME/foo/lib ; foo [args]
$ LD_LIBRARY_PATH=$HOME/foo/lib   foo [args]
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

3.9 Labs



Video Demonstration Resources

[using_renice_demo.mp4](#)

Exercise 3.1: Controlling Processes with ulimit

Please do:

```
$ help ulimit
```

and read [/etc/security/limits.conf](#) before doing the following steps.

1. Start a new shell by typing **bash** (or opening a new terminal) so that your changes are only effective in the new shell.
View the current limit on the number of open files and explicitly view the hard and soft limits.
2. Set the limit to the hard limit value and verify if it worked.
3. Set the hard limit to 2048 and verify it worked.
4. Try to set the limit back to the previous value. Did it work?

✓ Solution 3.1

```
1. $ bash  
$ ulimit -n
```

1 1024

```
$ ulimit -S -n
```

1 1024

```
$ ulimit -H -n
```

1 4096

```
2. $ ulimit -n hard  
$ ulimit -n
```

1 4096

```
3. $ ulimit -n 2048  
$ ulimit -n
```

1 2048

```
4. $ ulimit -n 4096
```

1 bash: ulimit: open files: cannot modify limit: Operation not permitted

```
$ ulimit -n
```

1 2048

For the exclusive use of LFS301 corp class taught 06 to 09 December

You can't do this anymore!

Note that if we had chosen a different limit, such as stack size (-s) we could raise back up again as the hard limit is unlimited.

Exercise 3.2: Examining System V IPC Activity

System V IPC is a rather old method of Inter Process Communication that dates back to the early days of **UNIX**. It involves three mechanisms:

1. Shared Memory Segments
2. Semaphores
3. Message Queues

More modern programs tend to use **POSIX IPC** methods for all three of these mechanisms, but there are still plenty of **System V IPC** applications found in the wild.

To get an overall summary of **System V IPC** activity on your system, do:

```
$ ipcs
```

```

1 ----- Message Queues -----
2 key      msqid      owner      perms      used-bytes     messages
3
4 ----- Shared Memory Segments -----
5 key      shmid      owner      perms      bytes      nattch      status
6 0x01114703 0      root       600        1000          6
7 0x00000000 98305    coop       600        4194304        2      dest
8 0x00000000 196610    coop       600        4194304        2      dest
9 0x00000000 23068675    coop       700        1138176        2      dest
10 0x00000000 23101444   coop       600        393216         2      dest
11 0x00000000 23134213   coop       600        524288         2      dest
12 0x00000000 24051718   coop       600        393216         2      dest
13 0x00000000 23756807   coop       600        524288         2      dest
14 0x00000000 24018952   coop       600        67108864        2      dest
15 0x00000000 23363593   coop       700        95408          2      dest
16 0x00000000 1441811    coop       600        2097152         2      dest
17
18 ----- Semaphore Arrays -----
19 key      semid      owner      perms      nsems
20 0x00000000 98304    apache     600        1
21 0x00000000 131073    apache     600        1
22 0x00000000 163842    apache     600        1
23 0x00000000 196611    apache     600        1
24 0x00000000 229380    apache     600        1

```

Note almost all of the currently running shared memory segments have a key of 0 (also known as `IPC_PRIVATE`) which means they are only shared between processes in a parent/child relationship. Furthermore, all but one are marked for destruction when there are no further attachments.

One can gain further information about the processes that have created the segments and last attached to them with:

```
$ ipcs -p
```

```

1 ----- Message Queues PIDs -----
2 msqid      owner      lpid      lrpid
3
4 ----- Shared Memory Creator/Last-op PIDs -----
5 shmid      owner      cpid      lpid
6 0          root      1023     1023

```

For the exclusive use of LFS301 corp class taught 06 to 09 December

7	98305	coop	2265	18780
8	196610	coop	2138	18775
9	23068675	coop	989	1663
10	23101444	coop	989	1663
11	23134213	coop	989	1663
12	24051718	coop	20573	1663
13	23756807	coop	10735	1663
14	24018952	coop	17875	1663
15	23363593	coop	989	1663
16	1441811	coop	2048	20573

Thus, by doing:

```
$ ps aux |grep -e 20573 -e 2048
```

1	coop	2048	5.3	3.7	1922996	305660	?	R1	Oct27	77:07	/usr/bin/gnome-shell
2	coop	20573	1.9	1.7	807944	141688	?	S1	09:56	0:01	/usr/lib64/thunderbird/thunderbird
3	coop	20710	0.0	0.0	112652	2312	pts/0	S+	09:57	0:00	grep --color=auto -e 20573 -e 2048

we see **thunderbird** is using a shared memory segment created by **gnome-shell**.

Perform these steps on your system and identify the various resources being used and by who. Are there any potential **leaks** (shared resources no longer being used by any active processes) on the system? For example, doing:

```
$ ipcs
```

1						
----- Shared Memory Segments -----							
3	key	shmid	owner	perms	bytes	nattch	
4						
5	0x00000000	622601	coop	600	2097152	2	
6	0x0000001a	13303818	coop	666	8196	0	
7						

shows a shared memory segment with no attachments and not marked for destruction. Thus it might persist forever, leaking memory if no subsequent process attaches to it.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Chapter 4

Signals



4.1	Signals	80
4.2	Types of Signals	81
4.3	kill	82
4.4	killall and pkill	83
4.5	Labs	85

4.1 Signals

What are Signals?

- Signals are generally sent by one of two methods
 - From the kernel to a user process as a result of an exception or programming error
 - From a user process (via system call) to the kernel which will send it to a user process
- Signals can only be sent between processes owned by the same user or from a process owned by the superuser to any process

Signals are one of the oldest methods of Inter-Process Communication (**IPC**) and are used to notify processes about **asynchronous** events (or **exceptions**).

By asynchronous, we mean the signal-receiving process may:

- Not expect the event to occur.
- Expect the event, but not know when it is most likely to occur.

For example, if a user decides to terminate a running program, it could send a signal to the process through the kernel to interrupt and kill the process.

When a process receives a signal, what it does will depend on the way the program is written. It can take specific actions, coded into the program, to handle the signal or it can just respond according to system defaults. Two signals:

- SIGKILL (#9)
- SIGSTOP (#19)

cannot be handled and will always terminate the program.

For the exclusive use of LFS301 corp class taught 06 to 09 December

4.2 Types of Signals

Types of Signals and Available Signals

- Exceptions detected by hardware (illegal memory reference, etc.)
- Exceptions such as process termination or sent by user
- `kill -l` shows all available signals:

```
student@openSUSE:~> kill -l
 1) SIGHUP      2) SIGINT      3) SIGQUIT      4) SIGILL      5) SIGTRAP
 6) SIGABRT    7) SIGBUS      8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV    12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN    22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS     34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
student@openSUSE:~>
```

Figure 4.1: Available Signals

There are a number of different types of signals and the signal sent by the kernel indicates what type of event (or exception) occurred. Generally, signals are used to handle two things, 1) exceptions detected by hardware (such as an illegal memory reference), or 2) exceptions generated by the environment (such as the premature death of a process from the user's terminal).

To see a list of the signals in **Linux**, along with their numbers use `kill -l` (that is the letter l not a one). The meaning attached to the signal type indicates what event occurred causing the signal to be sent to a process (when sent from the kernel).

The signals from `SIGRTMIN` on are termed **real-time signals** and are a relatively recent addition. They have no predefined purpose, and differ in some important ways from normal signals; they can be queued up and are handled in a **FIFO** (First In First Out) order.

The meaning attached to the signal type indicates what event caused the signal to be sent. While users can explicitly send any signal type to one of their processes, the meaning attached may no longer be implied by the signal number or type, and can be used in any way that the process desires.

Typing `man 7 signal` will give further documentation.

4.3 kill

kill

- Users invoke **kill** to send signals to other processes
- **kill** is a misnomer, should be send-signal
- `kill [pid]` sends the SIGTERM signal, by default
 - If process does not handle (or catch) or ignore SIGTERM, then process is killed
- `kill -signal [pid]` sends a specific signal to a process

Since a process cannot send a signal directly to another process, it must ask the kernel to send the signal. Users (including the superuser) can send signals to other processes (programs) by using **kill**.

kill is really a bad name. Although it is used a lot to kill processes, it is really designed to send signals to processes. Therefore, **kill** is not really accurate.

The default signal sent is SIGTERM (#15) which can be handled, or ignored by the receiving process in order to prevent its death. It is preferable to use this signal to give the process a chance to clean up after itself. If this signal is ignored, the user can usually send a SIGKILL (#9) signal, which cannot be ignored, to kill the process.

We use **kill** to send signals (by either number or name):

```
$ kill 1234  
$ kill -9 1234  
$ kill -SIGTERM 1234
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

4.4 killall and pkill

killall

- Uses process name instead of pid
- Sends signal to all processes with that name (assuming appropriate privileges)

```
$ killall bash  
$ killall -9 bash  
$ killall -SIGKILL bash
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

pkill

- Sends signals to process using name

```
$ pkill [-signal] [options] [pattern]
      – -P: ppid
      – -G: gid
      – -U: uid
```

Similar to **kill** but uses name instead of pid.

Make **rsyslog** reread its configuration file:

```
$ pkill -HUP rsyslogd
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

4.5 Labs

Exercise 4.1: Examining Signal Priorities and Execution

We give you a C program that includes a signal handler that can handle any signal. The handler avoids making any system calls (such as those that might occur while doing I/O). This file can be extracted from your downloaded SOLUTIONS file as `signals.c`.



signals.c

```

1  /*
2  * Examining Signal Priorities.
3  *
4  * In the below, do not send or handle either of the signals SIGKILL
5  * or SIGSTOP.
6  *
7  * Write a C program that includes a signal handler that can handle
8  * any signal. The handler should avoid making any system calls (such
9  * as those that might occur doing I/O).
10 *
11 * The handler should simply store the sequence of signals as they
12 * come in, and update a counter array for each signal that indicates
13 * how many times the signal has been handled.
14 *
15 * The program should begin by suspending processing of all signals
16 * (using sigprocmask()).
17 *
18 * It should then install the new set of signal handlers (which can be
19 * the same for all signals, registering them with the sigaction()
20 * interface.
21 *
22 * The the program should send every possible signal to itself multiple
23 * times, using the raise() function.
24 *
25 * Signal processing should be resumed, once again using sigprocmask().
26 *
27 * Before completing, the program should print out statistics
28 * including:
29 *
30 *     The total number of times each signal was received.
31 *
32 *     The order in which the signals were received, noting each time the total
33 *     number of times that signal had been received up to that point.
34 *
35 *
36 * Note the following items:
37 *
38 *     If more than one of a given signal is raised while the process
39 *     has blocked it, does the process receive it multiple times?
40 *
41 *     Are all signals received by the process, or are some handled
42 *     before they reach it?
43 *
44 *     What order are the signals received in?
45 *
46 *     If you are running KDE as your desktop environment, you may find
47 *     signal 32 blocked. One signal, SIGCONT (18 on x86) may not get
48 *     through; can you figure out why?
49 *
50 *     It appears that signals 32 and 33 can not be blocked and will cause

```

For the exclusive use of LFS301 corp class taught 06 to 09 December



```

51 * your program to fail. Even though header files indicate
52 * SIGRTMIN=32, the command kill -l indicates SIGRTMIN=34.
53 *
54 * Avoid sending these signals.
55 *
56 */
57
58 #include <stdio.h>
59 #include <unistd.h>
60 #include <signal.h>
61 #include <stdlib.h>
62 #include <string.h>
63 #include <pthread.h>
64
65 #define NUMSIGS 64
66
67 /* prototypes of locally-defined signal handlers */
68
69 void (sig_handler) (int);
70
71 int sig_count[NUMSIGS + 1];           /* counter for signals received */
72 volatile static int line;
73 volatile int signumbuf[6400], sigcountbuf[6400];
74
75 int main(int argc, char *argv[])
76 {
77     sigset_t sigmask_new, sigmask_old;
78     struct sigaction sigact, oldact;
79     int signum, rc, i;
80
81     /* block all possible signals */
82     rc = sigfillset(&sigmask_new);
83     rc = sigprocmask(SIG_SETMASK, &sigmask_new, &sigmask_old);
84
85     /* Assign values to members of sigaction structures */
86     memset(&sigact, 0, sizeof(struct sigaction));
87     sigact.sa_handler = sig_handler;          /* we use a pointer to a handler */
88     sigact.sa_flags = 0;                      /* no flags */
89     /* VERY IMPORTANT */
90     sigact.sa_mask = sigmask_new;            /* block signals in the handler itself */
91
92     /*
93      * Now, use sigaction to create references to local signal
94      * handlers * and raise the signal to myself
95      */
96
97     printf
98         ("\nInstalling signal handler and Raising signal for signal number:\n\n");
99     for (signum = 1; signum <= NUMSIGS; signum++) {
100         if (signum == SIGKILL || signum == SIGSTOP || signum == 32
101             || signum == 33) {
102             printf(" --");
103         } else {
104             sigaction(signum, &sigact, &oldact);
105             /* send the signal 3 times! */
106             rc = raise(signum);
107             rc = raise(signum);
108             rc = raise(signum);
109             if (rc) {
110                 printf("Failed on Signal %d\n", signum);

```

For the exclusive use of LFS301 corp class taught 06 to 09 December

```

111         } else {
112             printf("%4d", signum);
113         }
114     }
115     if (signum % 16 == 0)
116         printf("\n");
117 }
118 fflush(stdout);

119 /* restore original mask */
120 rc = sigprocmask(SIG_SETMASK, &sigmask_old, NULL);

121
122 printf("\nSignal Number(Times Processed)\n");
123 printf("-----\n");
124 for (i = 1; i <= NUMSIGS; i++) {
125     printf("%4d:%3d ", i, sig_count[i]);
126     if (i % 8 == 0)
127         printf("\n");
128 }
129 printf("\n");
130
131 printf("\nHistory: Signal Number(Count Processed)\n");
132 printf("-----\n");
133 for (i = 0; i < line; i++) {
134     if (i % 8 == 0)
135         printf("\n");
136     printf("%4d(%1d)", signumbuf[i], sigcountbuf[i]);
137 }
138 printf("\n");
139 exit(EXIT_SUCCESS);
140 }

141 }

142 void sig_handler(int sig)
143 {
144     sig_count[sig]++;
145     signumbuf[line] = sig;
146     sigcountbuf[line] = sig_count[sig];
147     line++;
148 }
```

You will need to compile it and run it as in:

```
$ gcc -o signals signals.c
$ ./signals
```

When run, the program:

- Does not send the signals SIGKILL or SIGSTOP, which can not be handled and will always terminate a program.
- Stores the sequence of signals as they come in, and updates a counter array for each signal that indicates how many times the signal has been handled.
- Begins by suspending processing of all signals and then installs a new set of signal handlers for all signals.
- Sends every possible signal to itself multiple times and then unblocks signal handling and the queued up signal handlers will be called.
- Prints out statistics including:
 - The total number of times each signal was received.

For the exclusive use of LFS301 corp class taught 06 to 09 December

- The order in which the signals were received, noting each time the total number of times that signal had been received up to that point.

Note the following:

- If more than one of a given signal is **raised** while the process has blocked it, does the process **receive** it multiple times? Does the behavior of **real time** signals differ from normal signals?
- Are all signals received by the process, or are some handled before they reach it?
- What order are the signals received in?

One signal, SIGCONT (18 on **x86**) may not get through; can you figure out why?



Please Note

On some **Linux** distributions signals 32 and 33 can not be blocked and will cause the program to fail. Even though system header files indicate SIGRTMIN=32, the command `kill -1` indicates SIGRTMIN=34.

Note that **POSIX** says one should use signal names, not numbers, which are allowed to be completely implementation dependent.

You should generally avoid sending these signals.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Chapter 5

Package Management Systems



5.1	Why Use Packages?	90
5.2	Software Packaging Concepts	91
5.3	Package Types	92
5.4	Available Package Management Systems	93
5.5	Packaging Tool Levels and Varieties	94
5.6	Package Sources	95
5.7	Creating Software Packages	96
5.8	Revision Control Systems	97
5.9	Available Source Control Systems	98
5.10	The Linux Kernel and git	99
5.11	Labs	101

5.1 Why Use Packages?

Why Use Packages?

- Automation: No need for manual installs and upgrades
- Scalability: Install packages on one system, or 10,000 systems
- Repeatability and predictability
- Security and auditing

Software package management systems are widely seen as one of the biggest advancements **Linux** brought to enterprise IT environments. By keeping track of files and metadata in an automated, predictable and reliable way, system administrators can use package management systems to make their installation processes scale to thousands of systems without requiring manual work on each individual system.

For the exclusive use of LFS301 corp class taught 06 to 09 December

5.2 Software Packaging Concepts

Software Packaging Concepts

- Gather and compress associated software files into a single package (archive), which may require other packages to be installed first
- Allow for easy software installation or removal
- Can verify file integrity via an internal database
- Can authenticate the origin of packages
- Facilitate upgrades
- Group packages by logical features
- Manage dependencies between packages

Package management systems supply the tools that allow system administrators to automate installing, upgrading, configuring and removing software packages in a known, predictable and consistent manner.

A given package may contain executable files, data files, documentation, installation scripts and configuration files. Also included are **metadata** attributes such as version numbers, checksums, vendor information, dependencies, descriptions, etc.

Upon installation, all that information is stored locally into an internal database which can be conveniently queried for version status and update information.

5.3 Package Types

Package Types

- **Binary** packages contain files ready for deployment, including executable files and libraries. These are architecture dependent.
- **Source** packages are used to generate binary packages; one should always be able to rebuild a binary package from the source package. One source package can be used for multiple architectures.
- **Architecture-independent** packages contain files and scripts that run under script interpreters, as well as documentation and configuration files.
- **Meta-packages** are groups of associated packages that collect everything needed to install a relatively large subsystem, such as a desktop environment, or an office suite etc.

Binary packages are the ones that system administrators have to deal with most of the time. On 64-bit systems that can run 32-bit programs, one may have two binary packages installed for a given program, perhaps one with **x86_64** or **amd64** in its name, and the other with **i386** or **i686** in its name.

Source packages can be helpful in keeping track of changes and source code used to come up with binary packages. They are usually not installed on a system by default but can always be retrieved from the vendor.

It should always be possible to rebuild binary packages from their source packages; for example on **RPM**-based systems one can rebuild the **p7zip** binary package by doing:

```
# rpmbuild --rebuild -rb p7zip-16.02-16.el8.src.rpm
```

which will place the results in `/root/rpmbuild`:

```
# root/rpmbuild>find . -name "*rpm"
1 ./RPMS/x86_64/p7zip-plugins-16.02-16.el8.x86_64.rpm
2 ./RPMS/x86_64/p7zip-debugsource-16.02-16.el8.x86_64.rpm
3 ./RPMS/x86_64/p7zip-plugins-debuginfo-16.02-16.el8.x86_64.rpm
4 ./RPMS/x86_64/p7zip-16.02-16.el8.x86_64.rpm
5 ./RPMS/noarch/p7zip-doc-16.02-16.el8.noarch.rpm
```

with the exact location depending on the **Linux** distribution and version.

For the exclusive use of LFS301 corp class taught 06 to 09 December

5.4 Available Package Management Systems

Available Package Management Systems



Figure 5.1: RPM and APT

Two very common package management systems:

- **RPM (Red Hat Package Manager)**

Used by all **Red Hat**-derived distributions, such as **Red Hat Enterprise Linux**, **Fedora**, **CentOS**, as well as by **SUSE** and its related community **openSUSE** distribution.

- **dpkg (Debian Package)**

Used by all **Debian**-derived distributions including **Debian**, **Ubuntu** and **Linux Mint**.

There are other package management systems, such as **portage/emerge** used by **Gentoo**, **pacman** used by **Arch**, and specialized ones used by **Embedded Linux** systems and **Android**.

Another ancient system is just to supply packages as **tarballs** without any real management or clean removal strategies; this approach still marks **Slackware**, one of the oldest **Linux** distributions.

But most of the time it is either **RPM** or **dpkg** and these are the only ones we will consider in this course.

For the exclusive use of LFS301 corp class taught 06 to 09 December

5.5 Packaging Tool Levels and Varieties

Packaging Tool Levels and Varieties

- **Low Level Utility:**

Install or remove a single package or a list of named ones

- Dependencies not fully handled, only warned about or cause error
- If another package needs to be installed first installation will fail
- If the package is needed by another package, removal will fail

- Examples: **rpm** and **dpkg**

- **High Level Utility**

Solves the dependency problems:

- Will install any other packages required
- Will avoid conflicts with already installed packages

- Examples: **dnf**, **yum**, **zypper**, **PackageKit**, and **apt**

There are two levels to packaging systems:

1. **Low Level Utility:** This simply installs or removes a single package, or a list of packages each one of which is individually and specifically named. Dependencies are not fully handled, only warned about or produce an error

- If another package needs to be installed first installation will fail.
- If the package is needed by another package, removal will fail.

The **rpm** and **dpkg** utilities play this role for the packaging systems that use them.

2. **High Level Utility:** This solves the dependency problems:

- If another package or group of packages needs to be installed before software can be installed, such needs will be satisfied.
- If removing a package interferes with another installed package, the administrator will be given the choice of either aborting, or removing all affected software.

The **dnf** and **zypper** utilities (and the older **yum**) take care of the dependency resolution for **rpm** systems, and **apt-get** and **apt-cache** and other utilities take care of it for **dpkg** systems.

In this course we will only discuss the command line interfaces to the packaging management systems; while the graphical front ends used by each **Linux** distribution can be useful, we would like to be less tied to any one interface and have more flexibility.

For the exclusive use of LFS301 corp class taught 06 to 09 December

5.6 Package Sources

Package Sources

- Distributions have multiple **repositories**
- All packages must work together well
- External repositories also exist; some interface cleanly, others cause conflicts

Every distribution has one or more package **repositories** where system utilities go to obtain software and to update with new versions. It is the job of the distribution to make sure all packages in the repositories play well with each other.

There are always other, external repositories, which can be added to the standard distribution-supported list. Sometimes these are closely associated with the distribution and only rarely produce significant problems; an example would be the **EPEL** (**E**xtra **P**ackages for **E**nterprise **L**inux) set of version-dependent repositories, which fit well with **RHEL** since their source is **Fedora** and the maintainers are close to **Red Hat**.

However, some external repositories are not very well constructed or maintained. For example, when a package is updated in the main repository, dependent packages may not be updated in the external one, which can lead to one form of **dependency hell**.

For the exclusive use of LFS301 corp class taught 06 to 09 December

5.7 Creating Software Packages

Creating Software Packages

- Create easy to install packages for custom software
- Packages allow for additional tasks to be performed during install
 - Creating needed symbolic links
 - Creating directories as needed
 - Setting permissions
 - Anything that can be scripted
- Many formats available
 - **RPM** for **Red Hat** and **SUSE**-based systems
 - **DEB** for **Debian** for **Debian**-based systems
 - **TGZ** for **Slackware**
 - **APK** for **Android**

Building your own custom software packages makes it easy to distribute and install your own software. Almost every version of **Linux** has some mechanism for doing this.

Building your own software allows you to control exactly what goes in the software and exactly how it is installed. You can create the package so that installing it runs scripts that perform all tasks needed to install the new software and/or remove the old software.

For the exclusive use of LFS301 corp class taught 06 to 09 December

5.8 Revision Control Systems

Why Use Revision Control Systems?

- Project complexity grows with size and number of contributors, users, and testers
- **Revision Control Systems (Version Control Systems):**
 - Keep accurate history (log) of changes
 - Can back up to earlier releases
 - Can resolve conflicting contributions
- Many methods in use

Software projects become more complex to manage as the size of the project increases, or as the number of developers working on them goes up.

In order to organize updates and facilitate cooperation, many different schemes are available for source control. Standard features of such programs included the ability to keep an accurate history, or log, of changes, be able to back up to earlier releases, coordinate possibly conflicting updates from more than one developer, etc.

Source Control Systems (or **Revision Control Systems**, as they are also commonly called) fill the role of coordinating cooperative development.

For the exclusive use of LFS301 corp class taught 06 to 09 December

5.9 Available Source Control Systems

Available Source Control Systems

Table 5.1: Available Source Control Systems

Product	URL
RCS	https://www.gnu.org/software/rcs/
CVS	https://www.nongnu.org/cvs
Subversion	https://subversion.apache.org
git	https://www.kernel.org/pub/software/scm/git/
GNU Arch	https://www.gnu.org/software/gnu-arch
Monotone	https://www.monotone.ca
Mercurial	https://www.mercurial-scm.org

There is no shortage of available products, both proprietary and open; Above is a brief list of products released under a **GPL** license.

We will focus only on **git**, a widely used product which arose from the **Linux** kernel development community. **git** has risen to a dominant position in use for open source projects in a remarkably short time, and is often used even in closed source environments

5.10 The Linux Kernel and git

The Linux Kernel and git

- The **Linux** kernel development system has special needs:
 - Widely distributed throughout the world
 - Thousands of developers
 - Very public, under **GPL** license
- **git** arose to satisfy those needs and has helped **Linux** to grow efficiently

The **Linux** kernel development system has special needs in that it is widely distributed throughout the world, with literally thousands of developers involved. Furthermore it is all done very publicly, under the **GPL** license.

For a long time there was no real source revision control system. Then major kernel developers went over to the use of **BitKeeper** (see <https://www.bitkeeper.com>), a commercial project which granted a restricted use license for **Linux** kernel development.

However in a very public dispute over licensing restrictions in the spring of 2005, the free use of **BitKeeper** became unavailable for **Linux** kernel development.

The response was the development of **git**, whose original author is Linus Torvalds. The source code for **git** can be obtained from <https://www.kernel.org/pub/software/scm/git/>, and full documentation can be found at <https://www.kernel.org/pub/software/scm/git/docs/>.

For the exclusive use of LFS301 corp class taught 06 to 09 December

How git Works

- Not a traditional version control system
- Does not work with **files** as basic unit
- The **Object Database** contains:
 - **Blobs**
 - **Trees**
 - **Commits**
- The **Directory Cache** captures state of the directory tree.
- Has graphical interfaces
- Sites such as **github** have millions of **git** repositories

Technically **git** is **not** a source control management system in the usual sense, and the basic units it works with are not files. It has two important data structures: an **object** database and a directory cache.

The object database contains objects of three varieties:

- **Blobs:** Chunks of binary data containing file contents.
- **Trees:** Sets of blobs including file names and attributes, giving the directory structure.
- **Commits:** Changesets describing tree snapshots.

The directory cache captures the state of the directory tree.

By liberating the controls system from a file by file based system, one is better able to handle changesets which involve many files.

git is always under rapid development and graphical interfaces to it are also under speedy construction. For example, see <https://www.kernel.org/git/>. One can easily browse particular changes as well as source trees.

Sites such as <https://www.github.com> now host literally millions of **git** repositories, both public and private. There are a host of easy to find articles, books, online tutorials, etc. on how to profitably use **git**.

For the exclusive use of LFS301 corp class taught 06 to 09 December

5.11 Labs



Video Demonstration Resources

[using_package_management_demo.mp4](#)

Exercise 5.1: Version Control with git

Making sure git is installed

Your system may already have **git** installed. Doing which **git** should show you if it is already present. If not, while you may obtain the source and compile and install it, it is usually easier to install the appropriate pre-compiled binary packages. Exact package names may vary, but one of the following should work depending on your distribution:

```
$ sudo apt-get install git* # Debian /Ubuntu
$ sudo zypper install git* # openSUSE
$ sudo dnf install git* # Fedora / RHEL 8 / CentOS 8
$ sudo yum install git* # RHEL 7 / CentOS 7
```

according to your particular distribution.

Let's get a feel for how **git works** and how easy it is to use. For now we will just make our own local project.

1. First we create a working directory and then initialize **git** to work with it:

```
$ mkdir git-test
$ cd git-test
$ git init
```

2. Initializing the project creates a **.git** directory which will contain all the version control information; the main directories included in the project remain untouched. The initial contents of this directory look like:

```
$ ls -l .git
1 total 40
2 drwxrwxr-x 2 coop coop 4096 Dec 30 13:59 branches/
3 -rw-rw-r-- 1 coop coop 92 Dec 30 13:59 config
4 -rw-rw-r-- 1 coop coop 58 Dec 30 13:59 description
5 -rw-rw-r-- 1 coop coop 23 Dec 30 13:59 HEAD
6 drwxrwxr-x 2 coop coop 4096 Dec 30 13:59 hooks/
7 drwxrwxr-x 2 coop coop 4096 Dec 30 13:59 info/
8 drwxrwxr-x 4 coop coop 4096 Dec 30 13:59 objects/
9 drwxrwxr-x 4 coop coop 4096 Dec 30 13:59 refs/
```

Later we will describe the contents of this directory and its subdirectories; for the most part they start out empty.

3. Next we create a file and add it to the project:

```
$ echo some junk > somejunkfile
$ git add somejunkfile
```

4. We can see the current status of our project with:

```
$ git status
1 On branch master
2
3 Initial commit
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

```

5 Changes to be committed:
6   (use "git rm --cached <file>..." to unstage)
7
8     new file: somejunkfile
9

```

Notice it is telling us that our file is **staged** but not yet **committed**.

- Let's tell **git** who is responsible for this repository:

```
$ git config user.name "Another Genius"
$ git config user.email "b_genius@linux.com"
```

This must be done for each new project unless you have it predefined in a global configuration file.

- Now let's modify the file, and then see the history of differences:

```
$ echo another line >> somejunkfile
$ git diff
```

```

1 diff --git a/somejunkfile b/somejunkfile
2 index 9638122..6023331 100644
3 --- a/somejunkfile
4 +++ b/somejunkfile
5 @@ -1 +1,2 @@
6   some junk
7 +another line

```

- To actually commit the changes to the repository we do:

```
$ git commit -m "My initial commit"
1 Created initial commit eafad66: My initial commit
2 1 files changed, 1 insertions(+), 0 deletions(-)
3 create mode 100644 somejunkfile
```

If you do not specify an identifying message to accompany the commit with the `-m` option you will jump into an editor to put some content in. You **must** do this or the commit will be rejected. The editor chosen will be what is set in your `EDITOR` environment variable, which can be superseded with setting `GIT_EDITOR`.

- You can see your history with:

```
$ git log
1 commit eafad66304ebbcd6acfe69843d246de3d8f6b9cc
2 Author: A Genius <a_genius@linux.com>
3 Date:   Wed Dec 30 11:07:19 2009 -0600
4
5   My initial commit
```

and you can see the information got in there. You will note the long hexadecimal string which is the **commit number**; it is a 160-bit, 40-digit unique identifier. **git** cares about these beasts, not file names.

- You are now free to modify the already existing file and add new files with `git add`. But they are staged until you do another `git commit`

- Now that was not so bad. But we have only scratched the surface.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Chapter 6

RPM



6.1	RPM (Red Hat Package Manager)	104
6.2	Package File Names	105
6.3	RPM Database and Helper Programs	106
6.4	Queries	107
6.5	Verifying Packages	108
6.6	Installing and Removing Packages	109
6.7	Updating, Upgrading and Freshening RPM Packages	111
6.8	Upgrading the Linux Kernel	113
6.9	rpm2archive and rpm2cpio	114
6.10	Labs	115

6.1 RPM (Red Hat Package Manager)

Advantages of Using RPM

- Manages software packages
 - Collection of files related to a single task or subsystem
 - Contains programs, data, documentation, and configuration information
 - May also contain dependency information
 - Does not retrieve packages
- Advantages for administrators
 - Easy to install and uninstall (erase) packages
 - Easy to verify that a package was installed correctly
 - Can use **FTP** or **HTTP** to install packages
- Advantages for developers
 - Can use original “pristine” sources
 - Can generate **RPMs** for different architectures

RPM is a package management utility developed by **Red Hat**. (The name originally stood for **Redhat Package Manager**.) All files related to a specific task are packaged into a single file, which also contains information about how and where to install and uninstall the files. When developers create a new version of a program, they usually release a new **RPM** package. Be aware that these files may not be usable for other **Linux** distributions.

For system administrators, **RPM** makes software packages easy to manage. It is easy to determine which package a particular file comes from, which version of the package is installed, and whether it was installed correctly. It is also easy to remove complete packages to free up disk space. **RPM** also distinguishes documentation files from the rest of a package, allowing you to choose whether to install documentation on a system.

RPM makes the software developer’s job easier. Often, developers must change the code to get it to compile and execute correctly on another operating system. **RPM** allows builders to keep the changes necessary for building on **Linux** separate from the original source. This capability facilitates incorporating new versions of the code, because build-related changes are all in one place. It also facilitates building versions of **Linux** for different architectures.



Installation from the Network

Unless given a specific URL to draw from, **rpm** does not retrieve packages over the network; it installs only from the local machine using absolute or relative paths.

However, one can also install directly from remote locations using protocols such as **HTTP**, or even the deprecated **FTP**.

For the exclusive use of LFS301 corp class taught 06 to 09 December

6.2 Package File Names

Package File Names

- Standard binary **RPM** package naming format:

```
<name>-<version>-<release>.<distro>.<architecture>.rpm  
sed-4.5-2.el8.x86_64.rpm
```

- Standard source **RPM** package naming format:

```
<name>-<version>-<release>.<distro>.src.rpm  
sed-4.5-2.el8.src.rpm
```

RPM package file names are based on fields that represent specific information, as documented in the **RPM** standard (<http://www.rpm.org/>)

Note that the `distro` field often actually specifies the repository that the package came from, as a given installation may use a number of different package repositories as we shall discuss when we discuss `dnf`, `yum` and `zypper` which work at a level above **RPM**.

For the exclusive use of LFS301 corp class taught 06 to 09 December

6.3 RPM Database and Helper Programs

RPM Database and Helper Programs

- **RPM** database located in `/var/lib/rpm`
- Database files stored as **Berkeley DB** hash files
- Other helper programs in this directory
- Database can be rebuilt with:
`$ sudo rpm --rebuilddb`

`/var/lib/rpm` is the default system directory which holds **RPM** database files in the form of **Berkeley DB** hash files. The database files should not be manually modified; updates should be done only through use of the **rpm** program.

An alternative database directory can be specified with the `--dbpath` option to the **rpm** program. One might do this, for example, to examine an **RPM** database copied from another system.

You can use the `--rebuilddb` option to rebuild the database indices from the installed package headers; this is more of a repair, and not a rebuild from scratch.

Helper programs and scripts used by **RPM** reside in `/usr/lib/rpm`. There are quite a few; for example on an **RHEL 8** system:

```
$ ls /usr/lib/rpm | wc -l
```

1 74

where **wc** is reporting the number of lines of output.

You can create an `rpmrc` file to specify default settings for **rpm**. By default, **rpm** looks for:

1. `/usr/lib/rpm/rpmrc`
2. `/etc/rpmrc`
3. `~/.rpmrc`

in the above order. Note all these files are read; **rpm** does not stop as soon as it finds that one exists.

An alternative `rpmrc` file can be specified using the `--rcfile` option.

For the exclusive use of LFS301 corp class taught 06 to 09 December

6.4 Queries

Queries

- All **rpm** queries include the **-q** option
- The **-q** option can be combined with numerous other query options:
 - f**: allows you to determine which package a file came from
 - l**: lists the contents of a specific package
 - a**: all the packages installed on the system
 - i**: information about the package
 - p**: run the query against a package file instead of the package database

Table 6.1: **rpm** Query Command Examples

Which version of a package is installed?	<code>rpm -q bash</code>
Which package did this file come from?	<code>rpm -qf /bin/bash</code>
What files were installed by this package?	<code>rpm -ql bash</code>
Show information about this package.	<code>rpm -qi bash</code>
Show information about this package from the package file, not the package data base.	<code>rpm -qip foo-1.0.0.1.noarch.rpm</code>
List all installed packages on this system.	<code>rpm -qa</code>

A couple of other useful parameters are **--requires** and **--whatprovides**.

The **--requires** option will return a list of prerequisites for a package, while the **--whatprovides** option will show what installed package provides a particular requisite package.

```
$ rpm -q --requires bash
$ rpm -qp --requires foo-1.0.0-1.noarch.rpm
$ rpm -q --whatprovides libc.so.6
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

6.5 Verifying Packages

Verifying Packages

- The `rpm -V` command is used to verify packages
- No output when everything is OK
 - S: filesize differs
 - M: mode differs (permissions and file type)
 - 5: MD5 sum differs
 - D: device major/minor number mismatch
 - L: readLink path mismatch
 - U: user ownership differs
 - G: group ownership differs
 - T: mTime differs
- To check all packages:

```
$ sudo rpm -Va
```

The `-V` option to `rpm` allows you to verify whether the files from a particular package are consistent with the system's RPM database. Use `rpm -Va` to verify all packages on the system.

In the output (you only see output if there is a problem) each of the characters denotes the result of a comparison of attribute(s) of the file to the value of those attribute(s) recorded in the database. A single "." (period) means the test passed, while a single "?" (question mark) indicates the test could not be performed (e.g. file permissions prevent reading). Otherwise, the character denotes the failure of the corresponding `--verify` test.

No output when everything is OK.

```
$ rpm -V bash
```

Output indicating that a file's size, checksum, and modification time have changed.

```
$ rpm -V logrotate
```

```
1 S.5....T. c /etc/logrotate.conf
```

Output indicating that a file is missing.

```
$ sudo mv /sbin/logrotate /sbin/logrotate_KEEP  
$ rpm -V logrotate
```

```
1 S.5....T. c /etc/logrotate.conf
```

```
2 missing /usr/sbin/logrotate
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

6.6 Installing and Removing Packages

Installing Packages

- RPM performs a number of tasks when installing a package
 - Dependency checks
 - Conflict checks
 - Commands required before installation
 - Handles configuration files with intelligent care
 - Unpacks files from package and installs them with correct attributes
 - Commands required after installation
 - Updates system **RPM** database
- rpm -i to install a package

```
$ sudo rpm -ivh bash-4.4.19-12.el8_0.x86_64
```

where the -i is for install, -v is for verbose, and -h means print hash marks to show progress

Dependency checks are necessary because some packages will not operate properly unless some other package is also installed.

Conflicts include attempts to install an already-installed package, or to install an older version over a newer version.

The person who builds a package can specify that certain tasks be performed before or after the install.

When installing a configuration file, if the file exists and has been changed since the previous version of the package was installed, **RPM** saves the old version with the suffix .rpmsave. This allows you to integrate the changes you have made to the old configuration file into the new version of the file. This feature depends on properly created RPM packages.

In addition to installing files in the right place, RPM also sets attributes such as permissions, ownership, and modification (build) time.

Every time **RPM** installs a package, it updates information in the system database. It uses this information when checking for conflicts.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Uninstalling Packages

- Successful uninstall normally produces no output

```
$ sudo rpm -e system-config-lvm
package system-config-lvm is not installed
```

- Error due to dependencies

```
$ sudo rpm --test -e xz
```

```
1 error: Failed dependencies:
2   xz is needed by (installed) pcp-5.1.1-4.el8_3.x86_64
3   xz is needed by (installed) sos-3.9.1-6.el8.noarch
4   xz is needed by (installed) gettext-devel-0.19.8.1-17.el8.x86_64
5   xz is needed by (installed) libreport-2.9.5-15.el8.x86_64
6   xz is needed by (installed) dracut-049-95.git20200804.el8_3.4.x86_64
7   xz is needed by (installed) libvirt-daemon-qemu-6.0.0-28.module+el8.3.0+7827+5e65edd7.x86_64
8   xz is needed by (installed) rpm-build-4.14.3-4.el8.x86_64
9   xz is needed by (installed) libguestfs-1:1.40.2-25.module+el8.3.0+7421+642fe24f.x86_64
10  xz is needed by (installed) sysstat-11.7.3-5.el8.x86_64
11  /usr/bin/xz is needed by (installed) file-roller-3.28.1-3.el8.x86_64
```

The `-e` option causes `rpm` to uninstall (erase) a package. Normally, `rpm -e` fails with an error message if the package you are attempting to uninstall is required by other packages on the system.

You can use the `--test` option along with `-e` to determine whether the uninstall would succeed or fail, without actually doing the uninstall. If the operation would be successful, `rpm` prints no output. Add the `-vv` option to get more information.

Remember the package argument for the erase is the package name, not the `rpm` file name.

IMPORTANT NOTE: Never remove (erase/uninstall) the `rpm` package itself. The only way to fix this problem is to re-install the OS, or with a rescue environment.

```
student@openSUSE:~$ sudo rpm -e xz
error: Failed dependencies:
      xz = 5.2.2 is needed by (installed) xz-lang-5.2.2-1.11.noarch
      xz is needed by (installed) logrotate-3.8.7-8.4.x86_64
      xz is needed by (installed) sysstat-11.0.6-2.4.x86_64
      xz is needed by (installed) zypper-log-1.13.21-5.3.1.noarch
      xz is needed by (installed) dracut-044-16.6.1.x86_64
```

Figure 6.1: Uninstalling RPM Packages

Note the example in the screenshot has somewhat different dependencies as it was done on an **openSUSE** system; the one on the slide was on a **RHEL** system.

For the exclusive use of LFS301 corp class taught 06 to 09 December

6.7 Updating, Upgrading and Freshening RPM Packages

Updating Packages

- Updating replaces original package, if installed
`$ rpm -Uvh bash-4.4.19-12.el8.x86_64.rpm`
- Installs package if not currently installed
- Cannot install two packages containing the same files.
- Can downgrade using --oldpackage option

When upgrading, the already installed package is removed after the newer version is installed. The one exception is the configuration files from the original installation which are kept with a .rpmsave extension.

If you use the -U option and the package is not already installed, it is simply installed and there is no error.

The -i option is not designed for upgrades; attempting to install a new **RPM** package over an older one fails with error messages because it tries to overwrite existing system files.

However different versions of the same package may be installed if each version of the package does not contain the same files: kernel packages and library packages from other architectures are typically the only packages that would be commonly installed multiple times.

If you want to downgrade with `rpm -U` (that is, to replace the current version with an earlier version), you must add the `--oldpackage` option to the command line.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Freshening Packages

- To **freshen** all packages in the current directory:

```
$ sudo rpm -Fvh *.rpm
```

- If an older version of a package is installed, it will be upgraded to the newer version in the directory.
- If the version on the system is the same as the one in the directory, nothing happens.
- If there is no version of a package installed, the package in the directory is ignored.
- Freshening can be useful for applying a lot of patches (i.e., upgraded packages) at once.

Freshening is similar to upgrading with one exception. When you are upgrading and the package does not already exist, then it is loaded. When you are freshening, if the package is not already loaded then it is simply ignored. Both upgrading and freshening will install a new package if the original package is already loaded.

The **-F** option is useful when you have downloaded several new patches and want to upgrade the packages that are already installed, but not install any new ones.

For the exclusive use of LFS301 corp class taught 06 to 09 December

6.8 Upgrading the Linux Kernel

Upgrading the Linux Kernel

- Always install a new kernel (do not upgrade)
On a **Red Hat**-based system:
`$ sudo rpm -ivh kernel-{version}.{arch}.rpm`
- Can revert to old kernel if there is problem
- Can remove old kernel after new one is tested, but it is always a good idea to keep at least one older kernel that is known to work

When you install a new kernel on your system, it requires a reboot (one of the few updates that do) to take effect. You should not do an upgrade (`-U`) of a kernel: an upgrade would remove the old currently running kernel.

This in and of itself will not stop the system, but if after a reboot you have any problems, you will no longer be able to reboot into the old kernel since it has been removed from the system. However, if you install (`-i`), both kernels coexist and you can choose to boot into either one; i.e., you can revert back to the old one if need be.

When you do this the **GRUB** configuration file will automatically be updated to include the new version; it will be the default choice at boot unless you reconfigure the system to do something else.

Once the new kernel version has been tested, you may remove the old version if you wish, though this is not necessary. Unless you are short on space, it is recommended that you keep one or more older kernels available.

6.9 rpm2archive and rpm2cpio

Using rpm2archive and rpm2cpio

- Convert an RPM package file to an archive:

```
$ rpm2archive bash-XXXX.rpm
```

creates `bash-XXXX.rpm.tgz`.

- Extract in one step:

```
$ cat bash-XXXX.rpm | rpm2archive - | tar -xvz
```

- Convert an RPM package file to a cpio archive:

```
$ rpm2cpio bash-XXXX.rpm > bash.cpio
```

- Extract one or more files:

```
$ rpm2cpio bash-XXXX.rpm | cpio -ivd bin/bash
```

```
$ rpm2cpio logrotate-XXXX.rpm | cpio --extract --make-directories
```

- List files in a package:

```
$ rpm -qilp bash-XXXX.rpm
```

rpm2archive is used to convert RPM package files to **tar** archives. If `-` is given as an argument, input and output will be on `stdin` and `stdout`.

It is more direct and efficient than the older **rpm2cpio** utility, which can be used to convert package files to **cpio** archives, or to extract files from the package file. When doing so, all files are extracted relative to the current directory. So if the user is in the `/home/student` directory and extracts the `bin/bash` file as in the example above, it will be stored in `/home/student/bin/bash`.

If all you want to do is list the files in a package, the easiest method is to use the **rpm** command itself:

```
$ rpm -qilp package.rpm
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

6.10 Labs



Video Demonstration Resources

[using_rpm_demo.mp4](#)

Exercise 6.0: RPM Required



On RedHat, CentOS, Fedora, or openSUSE

To do these labs you need to have access to a system that is **RPM**-based, such as **RHEL**, **CentOS**, **Fedora**, **SUSE**, or **openSUSE**.

Exercise 6.1: Using RPM

Here we will just do a number of simple operations for querying and verifying **rpm** packages.

This lab will work equally well on **Red Hat** and **SUSE**-based systems.

1. Find out what package the file `/etc/logrotate.conf` belongs to.
2. List information about the package including all the files it contains.
3. Verify the package installation.
4. Try to remove the package.

Solution 6.1

1. `$ rpm -qf /etc/logrotate.conf`

1 logrotate-3.14.0-3.el8.x86_64

2. `$ rpm -qil logrotate`

1 ...

Note a fancier form that combines these two steps would be:

`$ rpm -qil $(rpm -qf /etc/logrotate.conf)`

3. `$ rpm -V logrotate`

1 ...?..... /etc/cron.daily/logrotate
2 S.5....T. c /etc/logrotate.conf



On RedHat

4. On **RHEL 8**:

`$ sudo rpm -e logrotate`

```
1 error: Failed dependencies:
2   logrotate is needed by (installed) vsftpd-3.0.3-28.el8.x86_64
3   logrotate >= 3.5.2 is needed by (installed) rsyslog-8.37.0-13.el8.x86_64
```

For the exclusive use of LFS301 corp class taught 06 to 09 December



On openSUSE

On openSUSE-Leap 15.1:

```
$ sudo rpm -e logrotate

1 error: Failed dependencies:
2   logrotate is needed by (installed) xdm-1.1.11-lp151.13.2.x86_64
3   logrotate is needed by (installed) wpa_supplicant-2.6-lp151.4.4.x86_64
4   logrotate is needed by (installed) chrony-3.2-lp151.8.6.x86_64
5   logrotate is needed by (installed) net-snmp-5.7.3-lp151.7.5.x86_64
6   logrotate is needed by (installed) syslog-service-2.0-lp151.3.3.noarch
7   logrotate is needed by (installed) vsftpd-3.0.3-lp151.6.3.x86_64
8   logrotate is needed by (installed) libvirt-daemon-5.1.0-lp151.7.6.1.x86_64
9   logrotate is needed by (installed) iscsiui-0.7.8.2-lp151.13.6.1.x86_64
10  logrotate is needed by (installed) mcelog-1.60-lp151.2.3.1.x86_64
```

Note that the exact package dependency tree depends on both the distribution and choice of installed software.

Exercise 6.2: Rebuilding the RPM Database

There are conditions under which the **RPM** database stored in `/var/lib/rpm` can be corrupted. In this exercise we will construct a new one and verify its integrity.

This lab will work equally well on **Red Hat** and **SUSE**-based systems.

1. Backup the contents of `/var/lib/rpm` as the rebuild process will overwrite the contents. If you neglect to do this and something goes wrong you are in serious trouble.
2. Rebuild the data base.
3. Compare the new contents of the directory with the backed up contents; don't examine the actual file contents as they are binary data, but note the number and names of the files.
4. Get a listing of all **rpms** on the system. You may want to compare this list with one generated before you actually do the rebuild procedure. If the query command worked, your new database files should be fine.
5. Compare again the two directory contents. Do they have the same files now?
6. You could delete the backup (probably about 100 MB in size) but you may want to keep it around for a while to make sure your system is behaving properly before trashing it.

Solution 6.2

1.

```
$ cd /var/lib
$ sudo cp -a rpm rpm_BACKUP
```
 2.

```
$ sudo rpm --rebuilddb
```
 3.

```
$ ls -l rpm rpm_BACKUP
```
 4.

```
$ rpm -qa | tee /tmp/rpm-qa.output
```
 5.

```
$ ls -l rpm rpm_BACKUP
```
 6. Probably you should not do this until you are sure the system is fine!
- ```
$ sudo rm -rf rpm_BACKUP
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Chapter 7

## dpkg



|     |                                             |     |
|-----|---------------------------------------------|-----|
| 7.1 | DPKG (Debian Package) . . . . .             | 118 |
| 7.2 | Package File Names and Source . . . . .     | 119 |
| 7.3 | DPKG Queries . . . . .                      | 120 |
| 7.4 | Installing/Upgrading/Uninstalling . . . . . | 121 |
| 7.5 | Labs . . . . .                              | 122 |

## 7.1 DPKG (Debian Package)

### DPKG Essentials

- Used in all **Debian**-based **Linux** distributions, including **Ubuntu** and **Linux Mint**
- Like **rpm**, the **dpkg** utility does not retrieve packages or remedy dependency failures.
- Package files have a .deb suffix
- **DPKG** database resides `/var/lib/dpkg`

**DPKG (Debian Package)** is the packaging system used to install, remove, and manage software packages under **Debian Linux** and other distributions derived from it. Like **RPM** it is not designed to directly retrieve packages in day to day use, but to install and remove them locally.

Package files have a .deb suffix and the **DPKG** database resides in the `/var/lib/dpkg` directory.

Like **rpm**, the **dpkg** program has only a partial view of the universe: it knows only what is installed on the system, and whatever it is given on the command line, but knows nothing of the other available packages whether they are in some other directory on the system or out on the Internet. As such it will also fail if a dependency is not met, or if one tries to remove a package other installed packages need.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 7.2 Package File Names and Source

# Package File Names and Source

- Standard naming format for a binary package:

<name>\_<version>-<revision\_number>\_<architecture>.deb

as in:

```
logrotate_3.14.0-4_amd64.deb
logrotate_3.14.0-4ubuntu3_amd64.deb
```

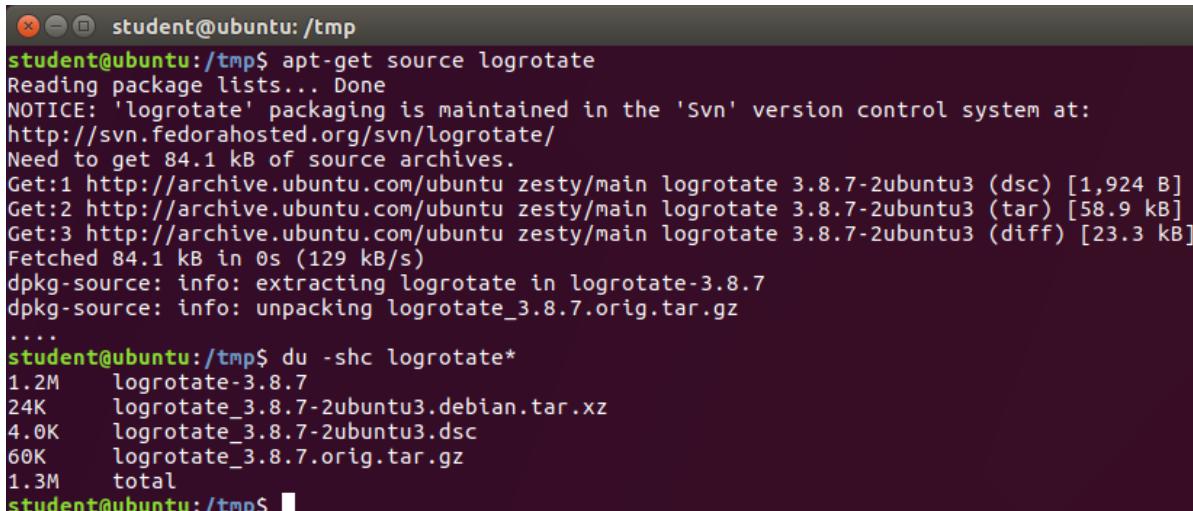
- Source package consists of at least 3 files:

- An **upstream tarball** (ending in .tar.gz)
- A **description file** (ending with .dsc)
- A second tarball (ending in .diff.gz or .debian.tar.gz) containing any patches to the upstream source, and additional files created for the package

**Debian** package file names are based on fields that represent specific information.

For historical reasons, the 64-bit **x86** platform is called **amd64** rather than **x86\_64**, and distributors such as **Ubuntu** manage to insert their name in the package name.

On an **Ubuntu** system one can download a source package and then see what files are downloaded or created as in:



```
student@ubuntu:/tmp$ apt-get source logrotate
Reading package lists... Done
NOTICE: 'logrotate' packaging is maintained in the 'Svn' version control system at:
http://svn.fedorahosted.org/svn/logrotate/
Need to get 84.1 kB of source archives.
Get:1 http://archive.ubuntu.com/ubuntu zesty/main logrotate 3.8.7-2ubuntu3 (dsc) [1,924 B]
Get:2 http://archive.ubuntu.com/ubuntu zesty/main logrotate 3.8.7-2ubuntu3 (tar) [58.9 kB]
Get:3 http://archive.ubuntu.com/ubuntu zesty/main logrotate 3.8.7-2ubuntu3 (diff) [23.3 kB]
Fetched 84.1 kB in 0s (129 kB/s)
dpkg-source: info: extracting logrotate in logrotate-3.8.7
dpkg-source: info: unpacking logrotate_3.8.7.orig.tar.gz
...
student@ubuntu:/tmp$ du -shc logrotate*
1.2M logrotate-3.8.7
24K logrotate_3.8.7-2ubuntu3.debian.tar.xz
4.0K logrotate_3.8.7-2ubuntu3.dsc
60K logrotate_3.8.7.orig.tar.gz
1.3M total
student@ubuntu:/tmp$
```

Figure 7.1: Getting Source Packages on Ubuntu

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 7.3 DPKG Queries

# DPKG Queries

- List all packages installed:

```
$ dpkg -l
```

- List files in the wget package:

```
$ dpkg -L wget
```

- Show info about an installed package, and a package file:

```
$ dpkg -s wget
$ dpkg -I webfs_1.21+ds1-8_amd64.deb
```

- List files in a package file:

```
$ dpkg -c webfs_1.21+ds1-8_amd64.deb
```

- Show what package owns /etc/init/networking.conf:

```
$ dpkg -S /etc/init/networking.conf
```

- Verify the installed package integrity

```
$ dpkg -V package
```

Only versions of **dpkg** greater than 1.17 support the **-V** option. Without arguments this will verify all packages on the system. See the **man** page to interpret the output.

## 7.4 Installing/Upgrading/Uninstalling

# Installing/Upgrading/Uninstalling

- Install or upgrade the **foobar** package:  
`$ sudo dpkg -i foobar.deb`
- Remove all of an installed package except for its configuration files:  
`$ sudo dpkg -r package`
- Remove all of an installed package including its configuration files.  
`$ sudo dpkg -P package`

When using the `-i` option (for install):

- If the package is not currently installed, then it will be installed.
- If the package is newer than the one currently installed then it will be upgraded.

Note the `-P` option stands for **purge**.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 7.5 Labs



### Video Demonstration Resources

[using\\_dpkg\\_demo.mp4](#)

## Exercise 7.0: dpkg Required



### On Debian, Ubuntu, or Linux Mint

To do these labs you need to have access to a system that is **Debian**-based, such as **Debian**, **Ubuntu**, or **Linux Mint**.

## Exercise 7.1: Using dpkg

Here we will just do a number of simple operations for querying and verifying **Debian** packages.

1. Find out what package the file `/etc/logrotate.conf` belongs to.
2. List information about the package including all the files it contains.
3. Verify the package installation.
4. Try to remove the package.

## Solution 7.1

1. `$ dpkg -S logrotate.conf`

```
1 logrotate: /usr/share/man/man5/logrotate.conf.5.gz
2 logrotate: /etc/logrotate.conf
3 rsync: /usr/share/doc/rsync/examples/logrotate.conf.rsync
```

2. `$ dpkg -L logrotate`

```
1 ...
```

3. `$ dpkg -V logrotate`

4. `$ sudo dpkg -r logrotate`

```
1 dpkg: dependency problems prevent removal of logrotate:
2 ubuntu-standard depends on logrotate; however:
3 Package logrotate is to be removed.
4 libvirt-daemon-system depends on logrotate.
5
6 dpkg: error processing package logrotate (--remove):
7 dependency problems - not removing
8 Errors were encountered while processing:
9 logrotate
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Chapter 8

## dnf and yum



|     |                                                  |     |
|-----|--------------------------------------------------|-----|
| 8.1 | Package Installers . . . . .                     | 124 |
| 8.2 | dnf and yum . . . . .                            | 125 |
| 8.3 | yum . . . . .                                    | 126 |
| 8.4 | Queries . . . . .                                | 127 |
| 8.5 | Installing/Removing/Upgrading Packages . . . . . | 128 |
| 8.6 | Additional dnf Commands . . . . .                | 129 |
| 8.7 | Labs . . . . .                                   | 130 |

## 8.1 Package Installers

# Package Installers

- Manage sources for packages to install
  - Local or remote **repositories**
- Used to automate install, upgrade, removal of software packages
- Resolve dependencies automatically
- Save time:
  - No need to download packages manually
  - No need to acquire dependency information

The lower level package **utilities** such as **rpm** and **dpkg** deal with the details of installing specific software package files and managing already installed software.

The higher level package **management systems** (such as **dnf**, **yum**, **apt** and **zypper**) work with databases of available software and incorporate the tools needed to find, install, update and uninstall software in a highly intelligent fashion.

The software repositories are provided by distributions and other independent software providers. The package installers maintain databases of available software derived from catalogs kept by the repositories. Unlike the low-level package tools, they have the ability to find and install dependencies automatically, a critical feature.

In this section we will talk about **dnf**, and **yum**; we will get to **zypper** and **apt** next.

## 8.2 dnf and yum

### What is dnf?

- Used as a front end to RPM
- Can fetch packages from remote repositories
  - Can rely on multiple independent repositories
  - Repository configuration files in `/etc/yum.repos.d/`
- Can resolve dependencies
- Used by **RHEL**, **CentOS**, **Fedor**a and others

```
student@c8stream:~$ ls /etc/yum.repos.d
CentOS-Stream-AppStream.repo CentOS-Stream-RealTime.repo
CentOS-Stream-BaseOS.repo epel-modular.repo
CentOS-Stream-Debuginfo.repo epel-playground.repo
CentOS-Stream-Extras.repo epel.repo
CentOS-Stream-HighAvailability.repo epel-testing-modular.repo
CentOS-Stream-Media.repo epel-testing.repo
CentOS-Stream-PowerTools.repo
[student@c8stream ~]$
```

Figure 8.1: rpm Repositories

**dnf** has a number of features that make it useful for package management. It is a front end to **RPM**, but also has the capabilities to retrieve packages from one or more remote repositories. One of its best features is the ability to resolve dependencies.

The configuration files for repositories are located in the `/etc/yum.repos.d` directory and have a `.repo` extension.

A very simple repo file might look like:



Sample Repo

```
[repo-name]
name=Description of the repository
baseurl=http://somesystem.com/path/to/repo
enabled=1
gpgcheck=1
```

**dnf** caches information and databases to speed up performance. To remove some or all cached information one can run the command:

```
$ dnf clean [packages | metadata | expire-cache | rpmbdb | plugins | all]
```

One can toggle use of a particular repo on or off by changing the value of `enabled`, or using the `--disablerepo somerepo` and `--enablerepo somerepo` options.

We will concentrate on the command line use of **dnf** and not consider the graphical interfaces distributions provide.

**For the exclusive use of LFS301 corp class taught 06 to 09 December**

## 8.3 yum

# yum

- **dnf** replaced **yum** during the **RHEL/CentOS 7** to **8** transition.
- **Fedora** has used it even longer
- **dnf** is backwards compatible:
  - Almost all common **yum** commands still work
- For more documentation, see:  
<https://docs.fedoraproject.org/en-US/quick-docs/dnf/>  
<https://dnf.readthedocs.io/en/latest/>

If you try to run **yum** commands, some versions of **dnf** will alert you that these are deprecated, and point you to the right command.

If you are an experienced **yum** user, you can gradually learn to use **dnf** because it accepts the subset of **yum** commands that take care of the majority of day to day tasks.

## 8.4 Queries

# Queries

- Can be used to search packages

```
$ dnf search
$ dnf info
$ dnf list
$ dnf grouplist
$ dnf groupinfo
```

- Can be used to search files

```
$ dnf provides
```

```
$ dnf search keyword
```

search for package with keyword in it.

```
$ dnf info package-name
```

Display information about a package.

```
$ dnf list [installed | updates | available]
```

List packages installed, available, or updates.

```
$ dnf grouplist
```

Show information about package groups installed, available, or updates.

```
$ dnf groupinfo packagegroup
```

Show information about a packagegroup.

```
$ dnf provides /path/to/file
```

Shows the owner of the package for file. (Note the need to use at least one / in the file name, which can be confusing.)

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 8.5 Installing/Removing/Upgrading Packages

# Installing/Removing/Upgrading Packages

- Commands:

```
$ sudo dnf install
$ sudo dnf localinstall
$ sudo dnf groupinstall
$ sudo dnf remove
$ sudo dnf update

- .rpmsave
- .rpmnew
```

```
$ sudo dnf install package
```

installs a package from a repository. Also resolves and installs dependencies.

```
$ sudo dnf localinstall package-file
```

installs a package from a local rpm file.

```
$ sudo dnf groupinstall 'group-name'
```

installs a specific software group from a repository. Also resolves and installs dependencies for each package in the group.

```
$ sudo dnf remove package
```

removes a package from the system.

```
$ sudo dnf update [package]
```

updates a package from a repository. If no package listed, updates all packages.

During installation (or update), if a package has a configuration file which is updated, it will rename the old configuration file with a .rpmsave extension. If the old configuration file will still work with the new software, it will name the new configuration file with a .rpmnew extension. You can search for these filename extensions to see if you need to do any reconciliation of the files.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 8.6 Additional dnf Commands

### Additional dnf Commands

```
$ sudo dnf repolist
$ sudo dnf shell
$ sudo dnf shell [text-file}
$ sudo dnf install --downloadonly
$ sudo dnf history
$ sudo dnf clean [packages|metadata|expire-cache|rpmdb|plugins|all]
```

```
$ sudo dnf list "dnf-plugin*"
```

lists additional **dnf** plugins

```
$ sudo dnf repolist
```

shows a list of enabled repositories.

```
$ sudo dnf shell
$ sudo dnf shell file.txt
```

provides an interactive shell in which to run multiple dnf commands. The second form executes the commands in `file.txt`.

```
$ sudo dnf install --downloadonly package
```

Will only download the packages for you. It stores them in the `/var/cache/dnf` directory.

```
$ sudo dnf history
```

View the history of **dnf** commands on the system, and with the correction options, even undo or redo previous commands.

```
$ sudo dnf clean [packages|metadata|expire-cache|rpmdb|plugins|all]
```

Clean up locally stored files and metadata under `/var/cache/dnf`. This saves space and does housecleaning of obsolete data.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 8.7 Labs

### Exercise 8.0: dnf or yum Required



#### On RedHat, CentOS, or Fedora

To do these labs you need to have access to a system that is **dnf**-based, such as **RHEL/CentOS** or **Fedora**. On **RHEL/CentOS 7** you should be able to substitute **yum** for **dnf** in the following commands and solution suggestions, as **dnf** is backwards compatible.

### Exercise 8.1: Basic dnf Commands

1. Check to see if there are any available updates for your system.
2. Update a particular package.
3. List all installed kernel-related packages, and list all installed or available ones.
4. Install the **httpd-devel** package, or anything else you might not have installed yet. Doing a simple:

```
$ sudo dnf list
```

will let you see a complete list; you may want to give a wildcard argument to narrow the list.

### Solution 8.1

1. 

```
$ sudo dnf update
$ sudo dnf check-update
$ sudo dnf list updates
```

Only the first form will try to do the installations.

2. 

```
$ sudo dnf update bash
```
3. 

```
$ sudo dnf list installed "kernel*"
$ sudo dnf list "kernel*"
```
4. 

```
$ sudo dnf install httpd-devel
```

### Exercise 8.2: Finding Information About a Package

Using **dnf** (and not **rpm** directly), find:

1. All packages that contain a reference to **bash** in their name or description.
2. Installed and available **bash** packages.
3. The package information for **bash**.
4. The dependencies for the **bash** package.

Try the commands you used above both as **root** and as a regular user. Do you notice any difference?

### Solution 8.2

For the exclusive use of LFS301 corp class taught 06 to 09 December

**Please Note**

Depending on your distribution version, you may get some permission errors if you do not use **sudo** with the following commands, even though we are just getting information.

```
$ sudo dnf search bash
$ sudo dnf list bash
$ sudo dnf info bash
$ sudo dnf deplist bash
```

**Exercise 8.3: Managing Groups of Packages****Please Note**

On **RHEL/CentOS** you may get some permission errors if you don't use **sudo** with some of the following commands, even when we are just getting information.

**dnf** provides the ability to manage groups of packages.

1. Use the following command to list all package groups available on your system:

```
$ dnf grouplist
```

2. Identify the Virtualization Host group and generate the information about this group using the command

```
$ dnf groupinfo "Virtualization Host"
```

3. Install using:

```
$ sudo dnf groupinstall "Virtualization Host"
```

4. Identify a package group that's currently installed on your system and that you don't need. Remove it using **dnf groupremove** as in:

```
$ sudo dnf groupremove "Virtualization Host"
```

Note you will be prompted to confirm removal so you can safely type the command to see how it works.

You may find that the **groupremove** does **not** remove everything that was installed; whether this is a bug or a feature can be discussed.

**Exercise 8.4: Adding a New Repository**

According to its authors (at <http://www.webmin.com/index.htm>):

**Webmin** is a web-based interface for system administration for Unix. Using any modern web browser, you can setup user accounts, Apache, DNS, file sharing and much more. Webmin removes the need to manually edit Unix configuration files like /etc/passwd, and lets you manage a system from the console or remotely."

We are going to create a repository for installation and upgrade. While we could simply go the download page and get the current **rpm**, that would not automatically give us any upgrades.

1. Create a new repository file called **webmin.repo** in the **/etc/yum.repos.d** directory. It should contain the following:



**webmin.repo**

```
[Webmin]
name=Webmin Distribution Neutral
baseurl=http://download.webmin.com/download/yum
```

For the exclusive use of LFS301 corp class taught 06 to 09 December



```
mirrorlist=http://download.webmin.com/download/yum/mirrorlist
enabled=1
gpgcheck=0
```

2. Install the webmin package.

```
$ sudo dnf install webmin
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Chapter 9

## zypper



|     |                                                  |     |
|-----|--------------------------------------------------|-----|
| 9.1 | zypper . . . . .                                 | 134 |
| 9.2 | Queries . . . . .                                | 135 |
| 9.3 | Installing/Removing/Upgrading Packages . . . . . | 136 |
| 9.4 | Additional zypper Commands . . . . .             | 137 |
| 9.5 | Labs . . . . .                                   | 138 |

## 9.1 zypper

# zypper

- Command-line tool for installing and managing packages in **SUSE Linux** and **openSUSE**
- Retrieves packages from a repository
- Resolves dependencies
- Works with **RPM** packages
- Is similar to **dnf** in commands and functionality

For use on **SUSE**-based systems, the **zypper** program provides a higher level of intelligent services for using the underlying **rpm** program, and plays the same role as **dnf/yum** on **Red Hat**-based systems. It can automatically resolve dependencies when installing, updating and removing packages. It accesses external software **repositories**, synchronizing with them and retrieving and installing software as needed.

**zypper** is the command line tool for installing and managing packages in **SUSE Linux** and **openSUSE**. It is very similar to **dnf** in its functionality and even in its basic command syntax, and also works with **rpm** packages.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 9.2 Queries

# Queries

- Commands

```
$ zypper list-updates
$ zypper repos
$ zypper search
$ zypper info
$ zypper search --provides /usr/bin/firefox
```

`$ zypper list-updates`

shows a list of available updates.

`$ zypper repos`

lists available repositories.

`$ zypper search <string>`

searches repositories for string.

`$ zypper info firefox`

lists info about a package.

`$ zypper search --provides /usr/bin/firefox`

searches repositories to show what packages provide a file.

## 9.3 Installing/Removing/Upgrading Packages

# Installing/Removing/Upgrading Packages

- zypper install  
    \$ sudo zypper install firefox
- zypper update  
    \$ sudo zypper --non-interactive update  
    \$ sudo zypper update firefox
- zypper remove  
    \$ sudo zypper remove firefox
- --non-interactive (or -n) does not ask for confirmation

```
$ sudo zypper install firefox
```

installs or updates package on the system.

```
$ sudo zypper --non-interactive install firefox
```

does not ask for confirmation of install or upgrade. This is useful for scripts.

```
$ sudo zypper update
```

updates all packages on system from a repository.

```
$ sudo zypper --non-interactive update
```

updates all packages on system from a repository, but does not ask for confirmation. This is useful for scripts.

```
$ sudo zypper remove firefox
```

removes a package from the system.

Like with **dnf**, one has to be careful with the removal command as any package that needs the package being removed would be removed as well.

---

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 9.4 Additional zypper Commands

### Additional zypper Commands

- Enter a **zypper** command line shell interface:

```
$ sudo zypper shell
```

- Add a new repository:

```
$ sudo zypper addrepo
```

- Remove a repository:

```
$ sudo zypper removerepo
```

- Clean up and save space in `/var/cache/zypp`:

```
$ sudo zypper clean [--all]
```

Sometimes a number of **zypper** commands must be run in a sequence. To avoid re-reading all the databases for each command, you can run **zypper** in **shell mode** as in:

```
$ sudo zypper shell
> install bash
...
> exit
```

Because **zypper** supports the **readline** library, you can use all the same command line editing functions in the **zypper** shell available in the **bash** shell.

To add a new repository:

```
$ sudo zypper addrepo URI alias
```

which is located at the supplied URI and will use the supplied alias.

To remove a repository from the list:

```
$ sudo zypper removerepo alias
```

using the alias of the repo you want to delete.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 9.5 Labs



### Video Demonstration Resources

`using_yast_demo.mp4`

## Exercise 9.0: zypper Required



### On openSUSE

To do these labs you need to have access to a system that is **zypper**-based, such as **SUSE**, or **openSUSE**.

## Exercise 9.1: Basic zypper Commands

1. Check to see if there are any available updates for your system.
2. Update a particular package.
3. List all repositories the system is aware of, enabled or not.
4. List all installed kernel-related packages, and list all installed or available ones.
5. Install the **apache2-devel** package, or anything else you might not have installed yet. (Note **httpd** is **apache2** on **SUSE** systems.) Doing a simple:

```
$ sudo zypper search
```

will let you see a complete list; you may want to give a wildcard argument to narrow the list.

## Solution 9.1

1. `$ zypper list-updates`
2. `$ sudo zypper update bash`
3. `$ zypper repos`
4. `$ zypper search -i kernel`  
`$ zypper search kernel`
5. `$ sudo zypper install apache2-devel`

## Exercise 9.2: Using zypper to Find Information About a Package

Using **zypper** (and not **rpm** directly), find:

1. All packages that contain a reference to **bash** in their name or description.
2. Installed and available **bash** packages.
3. The package information for **bash**.
4. The dependencies for the **bash** package.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Try the commands you used above both as `root` and as a regular user. Do you notice any difference?

## ✓ Solution 9.2

1. `$ zypper search -d bash`

Without the `-d` option only packages with `bash` in their actual name are reported. You may have to do `zypper info` on the package to see where `bash` is mentioned.

2. `$ zypper search bash`

3. `$ zypper info bash`

4. `$ zypper info --requires bash`

will give a list of files `bash` requires. Perhaps the easiest way to see what depends on having `bash` installed is to do

`$ sudo zypper remove --dry-run bash`

For this exercise `bash` is a bad choice since it is so integral to the system; you really can't remove it anyway.

---

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Chapter 10

## APT



|      |                                        |     |
|------|----------------------------------------|-----|
| 10.1 | APT                                    | 142 |
| 10.2 | APT Utilities                          | 143 |
| 10.3 | Queries                                | 144 |
| 10.4 | Installing/Removing/Upgrading Packages | 145 |
| 10.5 | Cleaning Up                            | 146 |
| 10.6 | Labs                                   | 147 |

## 10.1 APT

# What is APT?

- Stands for **Advanced Packaging Tool**
- Includes utilities such as **apt-get** and **apt-cache** (which invoke the lower level **dpkg** program)
- Works with **Debian** packages (.deb file extension)
- Not uncommon to use a repository on more than one **Debian-based Linux** distribution.



### apt vs apt-get

For almost all interactive purposes, it is no longer necessary to use **apt-get**; one can just use the shorter name **apt**. However, you will see **apt-get** used all the time out of habit, and it works better in scripts. In this course we will use **apt-get** because the commands we reference can be used either at the command line or in scripts.

For use on **Debian**-based systems, the **APT** (**Advanced Packaging Tool**) set of programs provides a higher level of intelligent services for using the underlying **dpkg** program, and plays the same role as **dnf** on **Red Hat**-based systems. The main utilities are **apt** and **apt-cache**. It can automatically resolve dependencies when installing, updating and removing packages. It accesses external software **repositories**, synchronizing with them and retrieving and installing software as needed.

Once again we are going to ignore graphical interfaces (on your computer) such as **synaptic** or the **Ubuntu Software Center**, or other older front ends to **APT** such as **aptitude**.

## 10.2 APT Utilities

### apt, apt-get, apt-cache, etc.

- Command-line tools for handling packages for **Debian Linux**
  - Install/manage individual packages
  - Upgrade packages
  - Upgrade distribution
- Front-ends to **dpkg**
- Also can retrieve packages from repositories
- Excellent resources for searching, examining and downloading packages:  
<https://www.debian.org/distrib/packages>  
<https://packages.ubuntu.com>

These utilities are the **APT** command line tools for package management. They can be used to install, manage and upgrade individual packages or the entire system, and can even upgrade the distribution to a completely new release, which can be a difficult task.

There are even (imperfect) extensions that let **apt** work with **rpm** files.

Like **dnf** and **zypper**, **APT** works with multiple remote repositories.

## 10.3 Queries

# Queries

- **apt-cache**

```
$ apt-cache search
$ apt-cache show
$ apt-cache showpkg
$ apt-cache depends
```

- **apt-file**

```
$ apt-file search
$ apt-file list
```

- You may have to install **apt-file** first and update its database as in:

```
$ sudo apt-get install apt-file
$ sudo apt-file update
```

```
$ apt-cache search apache2
```

Search the repository for a package named **apache2**.

```
$ apt-cache show apache2
```

Display basic information about the **apache2** package.

```
$ apt-cache showpkg apache2
```

Displays detailed information about the **apache2** package.

```
$ apt-cache depends apache2
```

List all dependent packages for **apache2**.

```
$ apt-file search apache2.conf
```

Search the repository for a file named **apache2.conf**.

```
$ apt-file list apache2
```

List all files in the **apache2** package.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 10.4 Installing/Removing/Upgrading Packages

# Installing/Removing/Upgrading Packages

- Install and remove packages:

```
$ sudo apt-get install glibc
$ sudo apt-get remove glibc
$ sudo apt-get --purge remove glibc
```

- **update** repository databases and then **upgrade** system or packages:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get dist-upgrade
```



### No individual package upgrades

Unlike with **dnf** or **zypper**, one never updates/upgrades individual packages; the entire package list is dealt with as a unit.

```
$ sudo apt-get install [package]
```

Used to install new packages or update a package which is already installed.

```
$ sudo apt-get remove [package]
```

Used to remove a package from the system. This does not remove the configuration files.

```
$ sudo apt-get --purge remove [package]
```

Used to remove a package and its configuration files from a system.

```
$ sudo apt-get update
```

Used to synchronize the package index files with their sources. The indexes of available packages are fetched from the location(s) specified in `/etc/apt/sources.list`.

```
$ sudo apt-get upgrade
$ sudo apt-get dist-upgrade
```

`upgrade` is used to apply all available updates to packages already installed. `dist-upgrade` will not update to a whole new distribution version as is commonly misunderstood.

Note that you must **update** before you **upgrade**, unlike with **dnf** or **zypper**, which can be confusing to habitual users of those utilities.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 10.5 Cleaning Up

# Cleaning Up

- Get rid of any packages not needed anymore, such as older **Linux** kernel versions:

```
$ sudo apt-get autoremove
```

- Clean out cache files and any archived package files that have been installed:

```
$ sudo apt-get clean
```

This can save a lot of space.

## 10.6 Labs



### Video Demonstration Resources

[using\\_package\\_gui\\_ubuntu\\_demo.mp4](#)

## Exercise 10.0: dpkg Required



### On Debian, Ubuntu, or Linux Mint

To do these labs you need to have access to a system that is **Debian**-based, such as **Debian**, **Ubuntu**, or **Linux Mint**.

## Exercise 10.1: Basic APT Commands

1. Check to see if there are any available updates for your system.
2. List all installed kernel-related packages, and list all installed or available ones.
3. Install the **apache2-dev** package, or anything else you might not have installed yet. Doing a simple:

```
$ apt-cache pkgnames
```

will let you see a complete list; you may want to give a wildcard argument to narrow the list.

## Solution 10.1

1. First synchronize the package index files with remote repositories:

```
$ sudo apt-get update
```

To actually upgrade:

```
$ sudo apt-get upgrade
$ sudo apt-get -u upgrade
```

(You can also use **dist-upgrade** as discussed earlier.) Only the first form will try to do the installations.

2. 

```
$ apt-cache search "kernel"
$ apt-cache search -n "kernel"
$ apt-cache pkgnames "kernel"
```

The second and third forms only find packages that have **kernel** in their name.

```
$ dpkg --get-selections "*kernel*"
```

to get only installed packages. Note that on **Debian**-based systems you probably should use **linux** not **kernel** for kernel-related packages as they don't usually have **kernel** in their name.

3. 

```
$ sudo apt-get install apache2-dev
```

## Exercise 10.2: Using APT to Find Information About a Package

Using **apt-cache** and **apt-get** (and not **dpkg**), find:

1. All packages that contain a reference to **bash** in their name or description.
2. Installed and available **bash** packages.

**For the exclusive use of LFS301 corp class taught 06 to 09 December**

3. The package information for **bash**.
4. The dependencies for the **bash** package.

Try the commands you used above both as `root` and as a regular user. Do you notice any difference?

## ✓ Solution 10.2

1. `$ apt-cache search bash`
2. `$ apt-cache search -n bash`
3. `$ apt-cache show bash`
4. `$ apt-cache depends bash`  
`$ apt-cache rdepends bash`

## ✍ Exercise 10.3: Managing Groups of Packages with APT

APT provides the ability to manage groups of packages, similarly to the way `dnf` does it, through the use of **metapackages**. These can be thought of as **virtual packages**, that collect related packages that must be installed and removed as a group.

To get a list of available **metapackages**:

```
$ apt-cache search metapackage
```

```
1 bacula - network backup service - metapackage
2 bacula-client - network backup service - client metapackage
3 bacula-server - network backup service - server metapackage
4 cloud-utils - metapackage for installation of upstream cloud-utils source
5 compiz - OpenGL window and compositing manager
6 emacs - GNU Emacs editor (metapackage)
7
```

You can then easily install them like regular single packages, as in:

```
$ sudo apt-get install bacula-client
1 Reading package lists... Done
2 Building dependency tree
3 Reading state information... Done
4 The following extra packages will be installed:
5 bacula-common bacula-console bacula-fd bacula-traymonitor
6 Suggested packages:
7 bacula-doc kde gnome-desktop-environment
8 The following NEW packages will be installed:
9 bacula-client bacula-common bacula-console bacula-fd bacula-traymonitor
10 0 upgraded, 5 newly installed, 0 to remove and 0 not upgraded.
11 Need to get 742 kB of archives.
12 After this operation, 1,965 kB of additional disk space will be used.
13 Do you want to continue? [Y/n]
```

Select an uninstalled metapackage and then remove it.

## Chapter 11

# System Monitoring



|      |                                         |     |
|------|-----------------------------------------|-----|
| 11.1 | System and Network Monitoring . . . . . | 150 |
| 11.2 | sar ** . . . . .                        | 152 |
| 11.3 | System Log Files . . . . .              | 154 |
| 11.4 | Labs . . . . .                          | 156 |



### Please Note

\*\* These sections may be considered in part or in whole as optional. They contain either background reference material, specialized topics, or advanced subjects. The instructor may choose to cover or not cover them depending on classroom experience and time constraints.

## 11.1 System and Network Monitoring

# Monitoring Tools

- Many command line utilities
  - Will show later for each subsystem
- Most draw information from `/proc` or `/sys`
- Graphical system monitors available on all **Linux** distributions:
  - **gnome-system-monitor**
  - **ksysguard**

While there are a number of graphical system monitors that hide many of the details, we will consider primarily the command line tools in this course.

**Linux** distributions come with many standard performance and profiling tools already installed. Many of them are familiar from other **UNIX**-like operating systems while some were developed specifically for **Linux**.

Most of these tools make use of mounted **pseudo-filesystems**, especially `/proc` and secondarily `/sys`, both of which we have already discussed when examining filesystems, and will revisit when we take up kernel configuration. We will look at them both.

The `/proc` and `/sys` pseudo-filesystems contain a lot of information about the system. Furthermore, many of the entries in these directory trees are writable and can be used to change system behavior; in most cases this requires a **root** user.

Like `/dev`, these are pseudo-filesystems because they exist totally in memory; if you look at the disk partition when the system is not running there will be only an empty directory which is used as a mount point.

Furthermore, the information displayed is gathered only when it is looked at; there is no constant or periodic polling to update entries.

# Network Monitoring Tools

Table 11.1: Network Monitoring Utilities

| Utility          | Purpose                                          | Package   |
|------------------|--------------------------------------------------|-----------|
| <b>netstat</b>   | Detailed networking statistics                   | netstat   |
| <b>iptraf</b>    | Gather information on network interfaces         | iptraf    |
| <b>tcpdump</b>   | Detailed analysis of network packets and traffic | tcpdump   |
| <b>wireshark</b> | Detailed network traffic analysis                | wireshark |

Given time constraints, it is not possible to include detailed discussion of network monitoring in this course. However, we do discuss network configuration in some detail later.

This topic is discussed in the course that is meant to follow this one: **LFS311: Linux for System Engineers**.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 11.2 sar \*\*

# sar

- **sar** stands for the **S**ystems **A**ctivity **R**eporter
- All-purpose tool for gathering data and creating reports
- **sadc** accumulates statistics and stores periodically in [/var/log/sa](#)
- **sar** is invoked via:  
`$ sar [ options ] [ interval ] [ count ]`  
and is repeated at `interval` seconds `count` times
- With no options gives a report on CPU usage

**sar** stands for the **S**ystems **A**ctivity **R**eporter. It is an all-purpose tool for gathering system activity and performance data and creating reports that are readable by humans.

On **Linux** systems the back end to **sar** is **sadc** (system activity data collector) which actually accumulates the statistics. It stores information in the [/var/log/sa](#) directory, with a daily frequency by default, but which can be adjusted. Data collection can be started from the command line, and regular periodic collection is usually started as a **cron** job stored in [/etc/cron.d/sysstat](#).

**sar** then reads in this data (either from the default locations or by use of a file specified with the `-f` option) and then produces a report.

```
student@ubuntu:~$ sudo sar 3 3
Linux 5.11.0 (ubuntu) 03/22/2021 _x86_64_ (4 CPU)

12:37:23 PM CPU %user %nice %system %iowait %steal %idle
12:37:26 PM all 0.08 0.00 0.00 0.08 0.00 99.83
12:37:29 PM all 0.00 0.00 0.00 0.00 0.00 100.00
12:37:32 PM all 0.00 0.00 0.00 0.00 0.00 100.00
Average: all 0.03 0.00 0.00 0.03 0.00 99.94
student@ubuntu:~$ |
```

Figure 11.1: Using sar

For the exclusive use of LFS301 corp class taught 06 to 09 December

## sar Example

- This example reports on paging, I/O and transfer data statistics

```
c8:/tmp># GETTING PAGING STATISTICS
c8:/tmp>sar -B 3 3
Linux 5.11.8 (c8) 03/22/2021 _x86_64_ (8 CPU)

02:49:22 PM pgpgin/s pgpgout/s fault/s majflt/s pgfree/s pgscank/s pgscand/s pgsteal/s %vmeff
02:49:25 PM 0.00 974.67 109.33 0.00 7483.67 0.00 0.00 0.00 0.00
02:49:28 PM 0.00 169.33 94.33 0.00 1268.00 0.00 0.00 0.00 0.00
02:49:31 PM 0.00 9.33 106.33 0.00 1095.00 0.00 0.00 0.00 0.00
Average: 0.00 384.44 103.33 0.00 3282.22 0.00 0.00 0.00 0.00
c8:/tmp># GETTING I/O AND TRANSFER RATE STATISTICS
c8:/tmp>sar -b 3 3
Linux 5.11.8 (c8) 03/22/2021 _x86_64_ (8 CPU)

02:50:09 PM tps rtps wtps bread/s bwrttn/s
02:50:12 PM 174.67 0.00 174.67 0.00 2002.67
02:50:15 PM 0.00 0.00 0.00 0.00 0.00
02:50:18 PM 1.00 0.00 1.00 0.00 13.33
Average: 58.56 0.00 58.56 0.00 672.00
c8:/tmp>
```

Figure 11.2: Using sar to Get I/O Statistics

Here is a list of the major **sar** options, or modes, each one of which has its own sub-options:

Table 11.2: sar Options

| Option | Meaning                                                                 |
|--------|-------------------------------------------------------------------------|
| -A     | Almost all information                                                  |
| -b     | I/O and transfer rate statistics (similar to <b>iostat</b> )            |
| -B     | Paging statistics including page faults                                 |
| -d     | Block device activity (similar to <b>iostat -x</b> )                    |
| -n     | Network statistics                                                      |
| -P     | Per CPU statistics (as in <b>sar -P ALL 3</b> )                         |
| -q     | Queue lengths (run queue, processes and threads)                        |
| -r     | Memory utilization statistics                                           |
| -S     | Swap utilization statistics                                             |
| -u     | CPU utilization (default)                                               |
| -v     | Statistics about inodes and files and file handles                      |
| -w     | Context switching statistics.                                           |
| -W     | Swapping statistics, pages in and out per second                        |
| -f     | Extract information from specified file, created by the -o option       |
| -o     | Save readings in the file specified, to be read in later with -f option |

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 11.3 System Log Files

# Log Files

- Most files are in `/var/log`
- Exact names vary with **Linux** distribution, e.g.:
  - `/var/log/messages` on **RHEL**
  - `/var/log/syslog` on **Ubuntu**

View continuously as new lines appear with:

```
$ sudo tail -f /var/log/messages
```

or

```
$ dmesg -w
```

which shows only kernel related messages

System log files are essential for monitoring and troubleshooting. In **Linux** these messages appear in various files under `/var/log`.

Ultimate control of how messages are dealt with is controlled by the **syslogd** (usually **rsyslogd** on modern systems) daemon, common to many **UNIX**-like operating systems. The newer **systemd**-based systems can use **journald** instead, but usually retain **syslogd** and cooperate with it.

Important messages are sent not only to the logging files, but also to the system **console** window; if you are not running a graphical interface or are at a virtual terminal, you will see them directly there as well. In addition these messages will be copied to `/var/log/messages` (or to `/var/log/syslog` on **Ubuntu**) but if you are running **X** or **Wayland** you have to take some steps to view them as they come in fresh.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Important Log Files

Here are some of the important log files found under `/var/log`:

Table 11.3: System Log Files

| File                                         | Purpose                                                                                                               |
|----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| <code>boot.log</code>                        | System boot messages                                                                                                  |
| <code>dmesg</code>                           | Kernel messages saved after boot. To see the current contents of the kernel message buffer, type <code>dmesg</code> . |
| <code>messages</code> or <code>syslog</code> | All important system messages                                                                                         |
| <code>secure</code>                          | Security related messages                                                                                             |

A good way to see log messages is to open a terminal window, and in that window type `tail -f /var/log/messages`. On a **GNOME** desktop you can also access the messages by clicking on **System->Administration->System Log** or **Applications->System Tools->Log File Viewer** in your Desktop menus, and other desktops have similar links you can locate.

In order to keep log files from growing without bound, the **logrotate** program is run periodically and keeps 4 previous copies (by default) of the log files (optionally compressed) and is controlled by `/etc/logrotate.conf`.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 11.4 Labs



### Video Demonstration Resources

[using\\_system\\_monitoring\\_demo.mp4](#)

#### Exercise 11.1: Using stress or stress-ng

**stress** is a C language program written by Amos Waterland at the University of Oklahoma, licensed under the **GPL v2**. It is designed to place a configurable amount of stress by generating various kinds of workloads on the system.

**stress-ng** is essentially an enhanced version of **stress**, which respects its symptoms and options. It is actively maintained: see <https://wiki.ubuntu.com/Kernel/Reference/stress-ng>

All major distributions should have **stress-ng** in their packaging systems. However, for **RHEL/CentOS** it needs to be obtained from the EPEL repository. As of this writing there is no package in the EPEL 8 repository, but you can install the one from EPEL 7 without a problem.

#### Installing from Source

If you are lucky you can install **stress** or **stress-ng** directly from your distribution's packaging system.

Otherwise, the source for **stress-ng** can be obtained using **git** from <https://kernel.ubuntu.com/git/cking/stress-ng.git/>. (Or you can download a tarball and use that.) To download, compile, and install:

```
$ git clone git://kernel.ubuntu.com/cking/stress-ng.git
$ cd stress-ng
$ make
$ sudo make install
```

Once installed, you can do:

```
$ stress-ng --help
```

for a quick list of options, or

```
$ info stress-ng
```

for more detailed documentation.

As an example, the command:

```
$ stress-ng -c 8 -i 4 -m 6 -t 20s
```

will:

- Fork off 8 CPU-intensive processes, each spinning on a `sqrt()` calculation.
- Fork off 4 I/O-intensive processes, each spinning on `sync()`.
- Fork off 6 memory-intensive processes, each spinning on `malloc()`, allocating 256 MB by default. The size can be changed as in `--vm-bytes 128M`.
- Run the stress test for 20 seconds.

After installing **stress-ng**, you may want to start up your system's graphical system monitor, which you can find on your application menu, or run from the command line, which is probably **gnome-system-monitor** or **ksysguard**.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Now begin to put stress on the system. The exact numbers you use will depend on your system's resources, such as the number of CPU's and **RAM** size.

For example, doing

```
$ stress-ng -m 4 -t 20s
```

puts only a memory stressor on the system.

Play with combinations of the switches and see how they impact each other. You may find the **stress-ng** program useful to simulate various high load conditions.

---

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 12

# Process Monitoring



|      |                              |     |
|------|------------------------------|-----|
| 12.1 | Process Monitoring . . . . . | 160 |
| 12.2 | ps . . . . .                 | 161 |
| 12.3 | pstree . . . . .             | 163 |
| 12.4 | top . . . . .                | 164 |
| 12.5 | Labs . . . . .               | 166 |

## 12.1 Process Monitoring

# Process Monitoring Tools

- **top**
- **uptime**
- **ps**
- **pstree**
- **mpstat**
- **iostat**
- **sar**
- **numastat**
- **strace**

To monitor processes, **Linux** administrators make use of many utilities, such as **ps**, **pstree** and **top**, all of which have long histories in **UNIX**-like operating systems.

Here is a list of some of the main tools for process monitoring:

Table 12.1: Process and Load Monitoring Tools

| Tool            | Purpose                                                         |
|-----------------|-----------------------------------------------------------------|
| <b>top</b>      | Process activity, dynamically updated                           |
| <b>uptime</b>   | How long the system is running and the average load             |
| <b>ps</b>       | Detailed information about processes                            |
| <b>pstree</b>   | A tree of processes and their connections                       |
| <b>mpstat</b>   | Multiple processor usage                                        |
| <b>iostat</b>   | CPU utilization and I/O statistics                              |
| <b>sar</b>      | Display and collect information about system activity           |
| <b>numastat</b> | Information about <b>NUMA</b> (Non-Uniform Memory Architecture) |
| <b>strace</b>   | Information about all system calls a process makes              |

The `/proc` filesystem can also be helpful in monitoring processes, as well as other items, on the system.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 12.2 ps

# Viewing Process States with ps

- Gets its information from `/proc`
- Extremely flexible in selection and presentation options
- Some common choices of options:

```
$ ps aux
$ ps -elf
$ ps -eL
$ ps -C "bash"
```

**ps** is a workhorse for displaying process characteristics and statistics, garnered from the `/proc` directory.

This command utility has existed in all **UNIX**-like operating system variants, and that diversity is reflected in the complicated potpourri of options that the **Linux** version of **ps** accepts, which fall into three categories:

1. **UNIX** options, which must be preceded by `-`, and which may be grouped.
2. **BSD** options, which must not be preceded by `-`, and which may be grouped.
3. **GNU** long options, each of which must be preceded by `--`.

Having all these possible options can make life rather confusing. Most system administrators tend to use one or two standard combinations for their daily use.

```
c8:/tmp>ps auxf
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
root 2 0.0 0.0 0 0 ? S 06:43 0:00 [kthreadd]
root 3 0.0 0.0 0 0 ? I< 06:43 0:00 _ [rcu_gp]
...
coop 3068 0.0 0.2 846052 38012 ?
coop 3103 0.0 0.0 237040 5648 pts/0 Ss 06:53 0:11 _ /usr/libexec/gnome-terminal-server
coop 51808 0.0 0.0 248280 6272 pts/0 S+ 13:41 0:00 | _ bash
coop 51809 0.0 0.0 219456 2496 pts/0 S+ 13:41 0:00 | _ git log
coop 3158 0.0 0.0 237512 5980 pts/1 Ss+ 06:53 0:00 | _ /usr/bin/less
c8:/tmp>
```

Figure 12.1: Using ps With BSD Options

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Customizing the ps Output

- Use the `-o` option, followed by a comma separated list of field identifiers to allow the user to print out a selected list of ps fields
  - pid: process id
  - uid: user id of process owner
  - cmd: command with all arguments
  - cputime: cumulative CPU time
  - pmem: ratio of the process's resident set size to the physical memory on the machine, expressed as a percentage
- See `ps` man page for many other output options

```
c8:/tmp>ps -o pid,user,uid,priority,cputime,pmem,size,command
 PID USER UID PRI TIME %MEM SIZE COMMAND
 47619 coop 1000 20 00:00:00 0.0 2656 bash
 49543 coop 1000 20 00:00:00 0.0 1100 ps -o pid,user,uid,priority,cputime,pmem,size,command
c8:/tmp>ps -o pid,user,uid,priority,cputime,pmem,size,command
```

Figure 12.2: Customizing ps Output

Another common options choice, `-elf` :

```
c8:/tmp>ps -elf
F S UID PID PPID C PRI NI ADDR SZ WCHAN STIME TTY TIME CMD
4 S root 1 0 0 80 0 - 60929 - 06:45 ? 00:00:02 /usr/lib/systemd/systemd --swit
ched-root --system --deserialize 18
1 S root 2 0 0 80 0 - 0 - 06:45 ? 00:00:00 [kthreadd]
1 I root 3 2 0 60 -20 - 0 - 06:45 ? 00:00:00 [rcu_gp]
...
0 S coop 6302 2365 1 80 0 - 182213 - 06:50 tty2 00:02:33 gnome-system-monitor
0 S coop 47591 37378 0 80 0 - 288155 - 07:55 pts/2 00:00:22 evince LFS301.pdf
0 S coop 47602 2089 0 80 0 - 53719 - 07:55 ? 00:00:00 /usr/libexec/evinced
...
0 S coop 54206 2715 0 80 0 - 58937 - 09:23 pts/4 00:00:00 bash
0 S root 54295 1140 0 80 0 - 54263 - 09:23 ? 00:00:00 sleep 60
4 R coop 54296 54206 0 80 0 - 67123 - 09:23 pts/4 00:00:00 ps -elf
c8:/tmp|
```

Figure 12.3: Using ps With UNIX Options

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 12.3 pstree

# Using pstree

- Shows the user the relationships between parent-child processes
- Use -p to show process IDs
- Use -H [pid] to highlight [pid] and its ancestors

```
$ pstree -aAps 31478
systemd,1 --switched-root --system --deserialize 21
`-vmplayer,31478
 |-{vmplayer},31570
 |-{vmplayer},31593
 |-{vmplayer},32572
 `-{vmplayer},32698
```

**pstree** gives a visual description of the process ancestry and multi-threaded applications:

```
$ pstree -aAp 2408
```

```
1 bash,2408
2 |-emacs,24998 pmonitor.tex
3 | |-{emacs},25002
4 | `-{emacs},25003
5 |-evince,18036 LFS201-SLIDES.pdf
6 | |-{evince},18040
7 | |-{evince},18046
8 | `-{evince},18047
```

Consult the **man** page for **pstree** for an explanation of many options; in the above we have chosen just to show information for pid=2408.

Note that one of its child processes (**evince**, pid=18036) has three children of its own. Another way to see that is:

```
$ ls -l /proc/18036/task
```

```
1 total 0
2 dr-xr-xr-x 5 coop coop 0 Sep 11 07:15 18036
3 dr-xr-xr-x 5 coop coop 0 Sep 11 07:15 18040
4 dr-xr-xr-x 5 coop coop 0 Sep 11 07:15 18046
5 dr-xr-xr-x 5 coop coop 0 Sep 11 07:15 18047
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 12.4 top

# top

- Used to display processes with highest CPU usage
- Processes are initially sorted by CPU usage
- If not run in secure mode (top s) user can signal processes
  - Press the **k** key
  - Give a PID when prompted
  - Give a signal number when prompted
- A ton of options, an ancient utility, lots of interactivity
- Hit **h** key to get list of key bindings
- Easy to control sorting options etc.

Press "h" for help. Display may be sorted by a number of different criteria. The display is refreshed every 5 seconds (though this is configurable) and continues to run until the user interrupts it (Ctrl-C).

| PID   | USER | PR | NI  | VIRT    | RES    | SHR    | S | %CPU | %MEM | TIME+   | COMMAND                 |
|-------|------|----|-----|---------|--------|--------|---|------|------|---------|-------------------------|
| 3605  | coop | 20 | 0   | 3793748 | 484588 | 123312 | S | 18.6 | 3.0  | 1:07.44 | thunderbird             |
| 2131  | coop | 20 | 0   | 1086964 | 83868  | 50788  | S | 3.3  | 0.5  | 8:21.00 | Xorg                    |
| 2107  | coop | 9  | -11 | 2729100 | 20408  | 13368  | S | 2.7  | 0.1  | 2:10.98 | pulseaudio              |
| 6302  | coop | 20 | 0   | 728788  | 49084  | 36336  | S | 1.7  | 0.3  | 2:52.34 | gnome-system-mo         |
| 44118 | coop | 20 | 0   | 3472692 | 219620 | 116372 | S | 1.3  | 1.4  | 0:55.83 | spotify                 |
| 2365  | coop | 20 | 0   | 4663756 | 432076 | 73556  | S | 1.0  | 2.7  | 7:15.12 | gnome-shell             |
| 3691  | coop | 20 | 0   | 36.8g   | 244560 | 89984  | S | 1.0  | 1.5  | 1:54.33 | slack                   |
| 50372 | coop | 20 | 0   | 1225676 | 67700  | 58684  | S | 0.7  | 0.4  | 0:25.04 | slack                   |
| 405   | root | 0  | -20 | 0       | 0      | 0      | I | 0.3  | 0.0  | 0:09.96 | kworker/u17:2-i915_flip |
| 2499  | root | 20 | 0   | 414964  | 31624  | 9704   | S | 0.3  | 0.2  | 0:08.90 | sssd_kcm                |
| 44149 | coop | 20 | 0   | 2183944 | 226052 | 78704  | S | 0.3  | 1.4  | 0:23.67 | spotify                 |
| 56244 | coop | 20 | 0   | 275384  | 5384   | 4544   | S | 0.3  | 0.0  | 0:00.21 | top                     |
| 1     | root | 20 | 0   | 243716  | 12144  | 8300   | S | 0.0  | 0.1  | 0:02.18 | systemd                 |

Figure 12.4: Using top

For the exclusive use of LFS301 corp class taught 06 to 09 December

## More on /proc

- The `/proc` filesystem is an interface to the kernel data structures
- `/proc` contains a subdirectory for each active process, named by the process id (PID)
- `/proc/self` is the currently executing process
- Some tunable parameters are in the `/proc` directories
- For more info, see the `proc` man page

In the sample output below, notice the sub-directories of the `/proc` filesystem which have numeric values for names. These ASCII numeric strings represent the PIDs of all active processes on the system. Each subdirectory contains additional files with information about that process.

```
student@opensuse:~/Desktop> ls /proc
1 2 280 36 482 652 8024 8184 8275 8502 driver loadavg sys
10 20 281 37 4854 66 8074 8196 8281 8514 dynamic_debug locks sysrq-trigger
1044 21 282 372 4881 67 8075 8201 8288 8520 execdomains mdstat sysvipc
1048 22 283 374 5 676 8077 8206 8291 8564 fb meminfo thread-self
1049 23 284 38 525 68 808 8212 8295 86 filesystems misc timer_list
11 24 29 39 528 69 8081 8216 8300 8606 fs modules tty
1127 25 291 4 530 7 8097 8221 8314 8654 interrupts mounts uptime
12 26 292 40 533 71 8113 8222 8316 9 iomem mpt version
123 262 295 41 564 72 8118 8224 8323 acpi ioports mtrr vmallocinfo
126 263 3 42 5677 73 8126 8225 8350 buddyinfo irq net vmstat
129 267 30 43 568 74 8136 8226 8351 bus kallsyms pagetypeinfo zoneinfo
13 268 3093 434 5955 75 8141 8227 8385 cgroups kcore partitions
14 27 31 453 6 7993 8146 8230 8387 cmdline keys sched_debug
145 275 32 454 602 8 8151 8231 8398 config.gz key-users schedstat
15 276 3258 457 62 8003 8155 8233 84 consoles kmsg scsi
16 277 33 461 63 8004 8160 8247 8431 cpuinfo kpagecgrou self
17 278 34 467 64 8014 8164 8255 8459 crypto kpagecount softirqs
18 279 35 471 65 8020 8166 8258 8469 devices kpageflags stat
19 28 355 4806 650 8022 8171 8271 85 diskstats latency_stats swaps
```

Figure 12.5: /proc Contents

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 12.5 Labs

### Exercise 12.1: Processes

1. Run **ps** with the options **-ef**. Then run it again with the options **aux**. Note the differences in the output.
2. Run **ps** so that only the process ID, priority, nice value, and the process command line are displayed.
3. Start a new **bash** session by typing **bash** at the command line. Start another **bash** session using the **nice** command but this time giving it a nice value of 10.
4. Run **ps** as in step 2 to note the differences in priority and nice values. Note the process ID of the two **bash** sessions.
5. Change the nice value of one of the **bash** sessions to 15 using **renice**. Once again, observe the change in priority and nice values.
6. Run **top** and watch the output as it changes. Hit q to stop the program.

### Solution 12.1

1. `$ ps -ef  
$ ps aux`

2. `$ ps -o pid,pri,ni,cmd`

```
1 PID PRI NI CMD
2 2389 19 0 bash
3 22079 19 0 ps -o pid,pri,ni,cmd
```

(Note: There should be no spaces between parameters.)

3. `$ bash  
$ nice -n 10 bash  
$ ps -o pid,pri,ni,cmd`

```
1 2389 19 0 bash
2 22115 19 0 bash
3 22171 9 10 bash
4 22227 9 10 ps -o pid,pri,ni,cmd
```

4. `$ renice 15 -p 22171  
$ ps -o pid,pri,ni,cmd`

```
1 PID PRI NI CMD
2 2389 19 0 bash
3 22115 19 0 bash
4 22171 4 15 bash
5 22246 4 15 ps -o pid,pri,ni,cmd
```

5. `$ top`

### Exercise 12.2: Monitoring Process States

1. Use **dd** to start a background process which reads from **/dev/urandom** and writes to **/dev/null**.
2. Check the process state. What should it be?
3. Bring the process to the foreground using the **fg** command. Then hit Ctrl-Z. What does this do? Look at the process state again, what is it?

**For the exclusive use of LFS301 corp class taught 06 to 09 December**

4. Run the **jobs** program. What does it tell you?
5. Bring the job back to the foreground, then terminate it using **kill** from another window.

## ✓ Solution 12.2

1. \$ dd if=/dev/urandom of=/dev/null &

2. \$ ps -C dd -o pid,cmd,stat

```
1 25899 dd if=/dev/urandom of=/dev/ R
```

Should be S or R.

3. \$ fg  
\$ ^Z  
\$ ps -C dd -o pid,cmd,stat

```
1 PID CMD STAT
2 25899 dd if=/dev/urandom of=/dev/ T
```

State should be T.

4. Type the **jobs** command. What does it tell you?

\$ jobs

```
1 [1]+ Stopped dd if=/dev/urandom of=/dev/null
```

5. Bring the job back to the foreground, then kill it using the **kill** command from another window.

\$ fg  
\$ kill 25899

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 13

# Memory Monitoring and Usage



|      |                                        |     |
|------|----------------------------------------|-----|
| 13.1 | Memory Monitoring and Tuning . . . . . | 170 |
| 13.2 | /proc/sys/vm . . . . .                 | 172 |
| 13.3 | vmstat . . . . .                       | 173 |
| 13.4 | Out of Memory Killer (OOM) . . . . .   | 174 |
| 13.5 | Labs . . . . .                         | 176 |

## 13.1 Memory Monitoring and Tuning

# Memory Monitoring

Important tools for monitoring and tuning memory in **Linux**

Table 13.1: **Memory Monitoring Utilities**

| Utility       | Purpose                                                               | Package |
|---------------|-----------------------------------------------------------------------|---------|
| <b>free</b>   | Brief summary of memory usage                                         | procps  |
| <b>vmstat</b> | Detailed virtual memory statistics and block I/O, dynamically updated | procps  |
| <b>pmap</b>   | Process memory map                                                    | procps  |

\$ free -m

|       | total | used | free | shared | buff/cache | available |
|-------|-------|------|------|--------|------------|-----------|
| Mem:  | 15901 | 2080 | 9317 | 992    | 4502       | 12457     |
| Swap: | 8095  | 0    | 8095 |        |            |           |

Over time, systems have become more demanding of memory resources at the same time RAM prices have decreased and performance has improved.

Yet, it is often the case that bottlenecks in overall system performance and throughput are memory-related; the CPUs and the I/O subsystem can be waiting for data to be retrieved from or written to memory.

There are many tools for monitoring, debugging and tuning a system's behavior with regard to its memory.

Tuning the memory sub-system can be a complex process. First of all, one has to take note that memory usage and I/O throughput are intrinsically related, as, in most cases, most memory is being used to cache the contents of files on disk.

Thus, changing memory parameters can have a large effect on I/O performance, and changing I/O parameters can have an equally large converse effect on the virtual memory sub-system.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# /proc/meminfo

```
$ cat /proc/meminfo
```

|                 |             |               |                |                    |            |
|-----------------|-------------|---------------|----------------|--------------------|------------|
| MemTotal:       | 16265960 kB | Writeback:    | 0 kB           | VmallocChunk:      | 0 kB       |
| MemFree:        | 10481080 kB | AnonPages:    | 2248272 kB     | Percpu:            | 6688 kB    |
| MemAvailable:   | 12596456 kB | Mapped:       | 811700 kB      | HardwareCorrupted: | 0 kB       |
| Buffers:        | 177824 kB   | Shmem:        | 659452 kB      | AnonHugePages:     | 716800 kB  |
| Cached:         | 2768600 kB  | KReclaimable: | 159792 kB      | ShmemHugePages:    | 0 kB       |
| SwapCached:     | 0 kB        | Slab:         | 307548 kB      | ShmemPmdMapped:    | 0 kB       |
| Active:         | 3320564 kB  | SRclaimable:  | 159792 kB      | FileHugePages:     | 0 kB       |
| Inactive:       | 1566504 kB  | SUnreclaim:   | 147756 kB      | FilePmdMapped:     | 0 kB       |
| Active(anon):   | 2381096 kB  | KernelStack:  | 15680 kB       | HugePages_Total:   | 0          |
| Inactive(anon): | 218792 kB   | PageTables:   | 74940 kB       | HugePages_Free:    | 0          |
| Active(file):   | 939468 kB   | NFS_Unstable: | 0 kB           | HugePages_Rsvd:    | 0          |
| Inactive(file): | 1347712 kB  | Bounce:       | 0 kB           | HugePages_Surp:    | 0          |
| Unevictable:    | 437372 kB   | WritebackTmp: | 0 kB           | Hugepagesize:      | 2048 kB    |
| Mlocked:        | 32 kB       | CommitLimit:  | 29713776 kB    | Hugetlb:           | 0 kB       |
| SwapTotal:      | 21580796 kB | Committed_AS: | 10239968 kB    | DirectMap4k:       | 258124 kB  |
| SwapFree:       | 21580796 kB | VmallocTotal: | 34359738367 kB | DirectMap2M:       | 9052160 kB |
| Dirty:          | 288 kB      | VmallocUsed:  | 24592 kB       | DirectMap1G:       | 8388608 kB |

The pseudofile `/proc/meminfo` contains a wealth of information about how memory is being used.

## 13.2 /proc/sys/vm

# /proc/sys/vm

- Contains many tunable knobs to control the Virtual Memory system
- Almost all of the entries are writable (by root)
- Exactly what appears in this directory will depend somewhat on the kernel version.
- Values can be changed either by directly writing to the entry, or using **sysctl**

When tweaking parameters in `/proc/sys/vm`, the usual best practice is to adjust one thing at a time and look for effects. The primary (inter-related) tasks are:

- Controlling flushing parameters; i.e., how many pages are allowed to be dirty and how often they are flushed out to disk.
- Controlling swap behavior; i.e., how much pages that reflect file contents are allowed to remain in memory, as opposed to those that need to be swapped out as they have no other backing store.
- Controlling how much memory overcommission is allowed, since many programs never need the full amount of memory they request, particularly because of copy on write (COW) techniques.

Memory tuning can be subtle: what works in one system situation or load may be far from optimal in other circumstances.

```
student@gentoo ~ $ ls /proc/sys/vm
admin_reserve_kbytes drop_caches mmap_min_addr page-cluster
block_dump extfrag_threshold mmap_rnd_bits panic_on_oom
compact_memory hugepages_treat_as_movable mmap_rnd_compat_bits percpu_pagelist_fraction
compact_unevictable_allowed hugetlb_shm_group nr_hugepages stat_interval
dirty_background_bytes laptop_mode nr_overcommit_hugepages stat_refresh
dirty_background_ratio legacy_va_layout nr_pdflush_threads swappiness
dirty_bytes lowmem_reserve_ratio oom_dump_tasks user_reserve_kbytes
dirty_expire_centisecs max_map_count oom_kill_allocating_task vfs_cache_pressure
dirty_ratio memory_failure_early_kill overcommit_kbytes watermark_scale_factor
dirtytime_expire_seconds memory_failure_recovery overcommit_memory
dirty_writeback_centisecs min_free_kbytes overcommit_ratio
```

Figure 13.1: /proc/sys/vm

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 13.3 vmstat

### vmstat

- Displays information about memory, paging, I/O, processor activity and processes.
- Many options
 

```
$ vmstat [options] [delay] [count]
```
- delay is given in seconds, report is repeated count times.
- First line shows averages since the last reboot
- Succeeding lines show activity during the specified interval.

```
File Edit View Search Terminal Help
c7:/tmp>vmstat 2 4
procs -----memory----- -----swap-----io-----system-----cpu-----
r b swpd free buff cache si so bi bo in cs us sy id wa st
2 0 0 3469116 887484 10275296 0 0 6504 53 393 147 4 7 85 4 0
1 0 0 3468316 887484 10275464 0 0 0 0 4820 593766 4 9 87 0 0
1 0 0 3468068 887484 10275464 0 0 0 20 3239 594743 4 10 87 0 0
1 0 0 3468068 887484 10275468 0 0 0 0 1621 599172 4 9 87 0 0
c7:/tmp>
```

Figure 13.2: **vmstat**

If the option **-S m** is given, memory statistics will be in MB instead of KB.

With the **-a** option, **vmstat** displays information about active and inactive memory pages: active pages are those recently used; they may be clean (disk contents are up to date) or dirty (need to be flushed to disk eventually). By contrast, inactive memory pages have not been recently used and are more likely to be clean and are released sooner under memory pressure: Memory moves back and forth between active and inactive lists, with new references, or a long time between uses.

```
File Edit View Search Terminal Help
c7:/tmp>vmstat -SM -a 2 4
procs -----memory----- -----swap-----io-----system-----cpu-----
r b swpd free inact active si so bi bo in cs us sy id wa st
2 0 0 6611 5972 2911 0 0 6450 52 392 311 4 7 85 4 0
2 0 0 6611 5972 2911 0 0 0 186 1602 601224 4 9 87 0 0
1 0 0 6612 5970 2911 0 0 0 0 1800 593070 4 9 86 0 0
1 0 0 6612 5970 2912 0 0 0 2 1615 587838 4 9 87 0 0
c7:/tmp>
```

Figure 13.3: **Using vmstat**

If you just want to get some quick statistics on only one partition, use the **-p** option

```
File Edit View Search Terminal Help
c7:/tmp>vmstat -p /dev/sdb1 2 4
sdb1 reads read sectors writes requested writes
358324 26917482 192161 3988152
358324 26917482 192161 3988152
358324 26917482 192161 3988152
358324 26917482 192161 3988152
c7:/tmp>
```

Figure 13.4: **Using vmstat on One Disk**

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 13.4 Out of Memory Killer (OOM)

### OOM Killer

- Systems may run out of physical memory. One can:
  1. Deny any further memory requests until memory is freed up
  2. Extend physical memory by the use of **swap** space
  3. Terminate (intelligently) selected processes to reduce memory usage and let the system survive
- **Linux** systems most often implement the second and third methods
- Which processes are terminated is selected by the **OOM** killer

The simplest way to deal with memory pressure would be to permit memory allocations to succeed as long as free memory is available and then fail when all memory is exhausted.

The second simplest way is to use swap space on disk to push some of the resident memory out of core; in this case, the total available memory (at least in theory) is the actual RAM plus the size of the swap space. The hard part of this is to figure out which pages of memory to swap out when pressure demands. In this approach, once the swap space itself is filled, requests for new memory must fail.

Linux, however, goes one better; it permits the system to overcommit memory, so that it can grant memory requests that exceed the size of RAM plus swap. While this might seem foolhardy, many (if not most) processes do not use all requested memory.

An example would be a program that allocates a 1 MB buffer, and then uses only a few pages of the memory. Another example is that every time a child process is forked, it receives a copy of the entire memory space of the parent. Because Linux uses the COW (copy on write) technique, unless one of the processes modifies memory, no actual copy needs be made. However, the kernel has to assume that the copy might need to be done.

Thus, the kernel permits overcommission of memory, but only for pages dedicated to user processes; pages used within the kernel are not swappable and are always allocated at request time.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# OOM Killer Algorithms

- Heuristic algorithm is not intended to be depended on for normal operations. Is there more for a graceful shutdown or retrenchment
- Process selection depends on a **badness** value which can be read from `/proc/[pid]/oom_score` for each process
- Adjustments can be made to a process's `oom_adj_score` in the same directory for each task.

One can modify, and even turn off overcommit by setting the value of `/proc/sys/vm/overcommit_memory`:

- 0: (default) Permit overcommit, but refuse obvious overcommits, and give root users somewhat more memory allocation than normal users.
- 1: All memory requests are allowed to overcommit.
- 2: Turn off overcommit. Memory requests will fail when the total memory commit reaches the size of the swap space plus a configurable percentage (50 by default) of RAM. This factor is modified changing `/proc/sys/vm/overcommit_ratio`.

An amusing take on this was given by Andries Brouwer (<https://lwn.net/Articles/104185/>):

An aircraft company discovered that it was cheaper to fly its planes with less fuel on board. The planes would be lighter and use less fuel and money was saved. On rare occasions however the amount of fuel was insufficient, and the plane would crash. This problem was solved by the engineers of the company by the development of a special OOF (out-of-fuel) mechanism. In emergency cases a passenger was selected and thrown out of the plane. (When necessary, the procedure was repeated.) A large body of theory was developed and many publications were devoted to the problem of properly selecting the victim to be ejected. Should the victim be chosen at random? Or should one choose the heaviest person? Or the oldest? Should passengers pay in order not to be ejected, so that the victim would be the poorest on board? And if for example the heaviest person was chosen, should there be a special exception in case that was the pilot? Should first class passengers be exempted? Now that the OOF mechanism existed, it would be activated every now and then, and eject passengers even when there was no fuel shortage. The engineers are still studying precisely how this malfunction is caused.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 13.5 Labs

### Exercise 13.1: Invoking the OOM Killer

Examine what swap partitions and files are present on your system by examining `/proc/swaps`.

Turn off all swap with the command

```
$ sudo /sbin/swapoff -a
```

Make sure you turn it back on later, when we are done, with

```
$ sudo /sbin/swapon -a
```

Now we are going to put the system under increasing memory pressure. One way to do this is to exploit the **stress-ng** program we installed earlier, running it with arguments such as:

```
$ stress-ng -m 12 -t 10s
```

which would keep 3 GB busy for 10 seconds.

You should see the **OOM** (Out of Memory) killer swoop in and try to kill processes in a struggle to stay alive. You can see what is going on by running **dmesg** or monitoring `/var/log/messages` or `/var/log/syslog`, or through graphical interfaces that expose the system logs.

Who gets clobbered first?

## Chapter 14

# I/O Monitoring and Tuning



|      |                |     |
|------|----------------|-----|
| 14.1 | I/O Monitoring | 178 |
| 14.2 | iostat         | 179 |
| 14.3 | iotop          | 180 |
| 14.4 | ionice **      | 181 |
| 14.5 | Labs           | 182 |



### Please Note

\*\* These sections may be considered in part or in whole as optional. They contain either background reference material, specialized topics, or advanced subjects. The instructor may choose to cover or not cover them depending on classroom experience and time constraints.

## 14.1 I/O Monitoring

# I/O Monitoring and Disk Bottlenecks

- Monitoring Input/Output (I/O) activity is essential to obtain peak performance
- Systems can be I/O bound, CPU bound, or memory bound; Only measurements can distinguish
- Three important tools:
  - **iostat**
  - **iotop**
  - **ionice**

Disk performance problems can be strongly coupled to other factors, such as insufficient memory or inadequate network hardware and tuning. Disentangling can be difficult.

As a rule, a system can be considered as I/O-bound when the CPU is found sitting idle waiting for I/O to complete, or the network is waiting to clear buffers.

However, one can be misled. What appears to be insufficient memory can result from too slow I/O; if memory buffers that are being used for reading and writing fill up, it may appear that memory is the problem, when the real problem is that buffers are not filling up or emptying out fast enough. Similarly, network transfers may be waiting for I/O to complete and cause network throughput to suffer.

Both real-time monitoring and tracing are necessary tools for locating and mitigating disk bottlenecks. However, rare or non-repeating problems can make this difficult to accomplish.

There are many relevant variables and I/O tuning is complex. We will also consider **I/O scheduling** later.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 14.2 iostat

### iostat

- Basic utility for monitoring all I/O activity
- Very flexible report options

```
$ iostat [OPTIONS] [devices] [interval] [count]
```

```
c8:/tmp>iostat
Linux 5.3.1 (c8) 09/30/2019 _x86_64_ (8 CPU)

avg-cpu: %user %nice %system %iowait %steal %idle
 2.09 0.01 0.33 0.18 0.00 97.39

Device tps kB_read/s kB_wrtn/s kB_read kB_wrtn
sda 0.44 242.46 0.25 5452482 5632
sdb 6.73 84.91 86.80 1909413 1952028
sdc 3.17 66.42 212.01 1493574 4767684
dm-0 0.17 28.37 0.31 638049 6892
dm-1 0.22 213.06 0.00 4791481 60
dm-2 0.01 0.15 0.00 3357 12
dm-3 0.01 0.16 0.00 3545 12
dm-4 0.00 0.06 0.00 1404 0
dm-5 0.00 0.00 0.00 82 0
dm-6 0.01 0.15 0.00 3412 12
dm-7 0.00 0.00 0.00 82 0
dm-8 0.00 0.06 0.00 1404 0
loop0 41.24 41.29 0.00 928544 0

c8:/tmp>
```

Figure 14.1: iostat

**iostat** is the basic workhorse utility for monitoring I/O device activity on the system. It can generate reports with a lot of information, with the precise content controlled by options.

After a brief summary of CPU utilization, I/O statistics are given: tps (I/O transactions per second; logical requests can be merged into one actual request), blocks read and written per unit time, where the blocks are generally sectors of 512 bytes; and the total blocks read and written.

Information is broken out by disk partition (and if **LVM** is being used also by dm (device mapper) logical partitions).

A more detailed report:

```
c8:/tmp>iostat -xk
Linux 5.3.1 (c8) 09/30/2019 _x86_64_ (8 CPU)

avg-cpu: %user %nice %system %iowait %steal %idle
 2.20 0.01 0.34 0.18 0.00 97.28

Device r/s w/s rkB/s wkB/s rrqm/s wrqm/s %rrqm %wrqm r_await w_await aqu-sz rreq-sz wareq-sz svctm %util
sda 0.35 0.08 232.35 0.25 0.00 0.02 0.99 16.41 73.69 127.10 0.04 662.98 3.08 1.07 0.05
sdb 2.33 5.30 81.40 90.98 0.02 7.65 0.86 59.66 0.30 1.03 0.00 34.90 17.17 0.38 0.29
sdc 1.45 1.65 67.78 202.40 0.08 1.94 5.28 54.02 1.04 13.65 0.02 46.75 122.37 0.38 0.09
dm-0 0.09 0.08 28.66 0.31 0.00 0.00 0.00 165.62 131.17 0.03 304.18 3.99 1.10 0.02
dm-1 0.21 0.00 282.69 0.00 0.00 0.00 0.00 165.62 131.17 0.03 304.18 3.99 1.10 0.02
dm-2 0.01 0.00 0.14 0.00 0.00 0.00 0.00 19.50 11.89 0.00 17.86 1.33 0.48 0.00
dm-3 0.01 0.00 0.15 0.00 0.00 0.00 0.00 19.50 20.00 0.00 18.56 1.33 0.42 0.00
dm-4 0.00 0.00 0.06 0.00 0.00 0.00 0.00 36.25 0.00 0.00 22.29 0.00 0.82 0.00
dm-5 0.00 0.00 0.00 0.00 0.00 0.00 0.00 50.36 0.00 0.00 2.93 0.00 0.86 0.00
dm-6 0.02 0.00 0.18 0.00 0.00 0.00 0.00 118.46 18.33 0.00 10.69 1.33 0.27 0.00
dm-7 0.00 0.00 0.00 0.00 0.00 0.00 0.00 78.43 0.00 0.00 2.93 0.00 0.79 0.00
dm-8 0.00 0.00 0.06 0.00 0.00 0.00 0.00 49.84 0.00 0.00 22.29 0.00 0.78 0.00
loop0 39.27 0.00 39.31 0.00 0.00 0.00 0.00 0.06 0.00 0.00 1.00 0.00 0.01 0.06
```

Figure 14.2: Using iostat

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 14.3 iotop

# iotop

- **iotop** displays current I/O activity
- Updates periodically like **top**
- Using the **-o** option can be useful to avoid clutter.

| File              | Edit       | View               | Search     | Terminal   | Help                                                                |
|-------------------|------------|--------------------|------------|------------|---------------------------------------------------------------------|
| Total DISK READ : | 116.67 M/s | Total DISK WRITE : | 132.23 K/s |            |                                                                     |
| Actual DISK READ: | 116.67 M/s | Actual DISK WRITE: | 0.00 B/s   |            |                                                                     |
| TID               | PRIOR      | USER               | DISK READ  | DISK WRITE | SWAPIN                                                              |
|                   |            |                    |            |            | IO> COMMAND                                                         |
| 17932             | be/4       | root               | 0.00 B/s   | 0.00 B/s   | 0.00 % 99.99 % [kworker/2:0]                                        |
| 3571              | be/4       | coop               | 0.00 B/s   | 0.00 B/s   | 0.00 % 99.99 % skype-bin                                            |
| 15601             | be/4       | coop               | 0.00 B/s   | 81.67 K/s  | 0.00 % 99.99 % vmware-vmx -ssnapshot.num-ntu-17-04.vmx [vmx-vcpu-3] |
| 19800             | be/4       | coop               | 0.00 B/s   | 0.00 B/s   | 0.00 % 99.99 % gnome-screenshot -i -w                               |
| 17186             | be/4       | coop               | 0.00 B/s   | 46.67 K/s  | 0.00 % 0.00 % vmware-vmx -ssnapshot.num-17-04.vmx [vmx-vthread-16]  |
| 15598             | be/4       | coop               | 0.00 B/s   | 3.89 K/s   | 0.00 % 0.00 % vmware-vmx -ssnapshot.num-ntu-17-04.vmx [vmx-vcpu-0]  |
| 19782             | be/4       | root               | 116.67 M/s | 0.00 B/s   | 0.00 % 0.00 % cat ./VIRTUAL/FC-25-LATEX/FC-25.vmdk                  |
| 1                 | be/4       | root               | 0.00 B/s   | 0.00 B/s   | 0.00 % 0.00 % systemd --switched-root --system --deserialize 21     |
| 2                 | be/4       | root               | 0.00 B/s   | 0.00 B/s   | 0.00 % 0.00 % [kthreadd]                                            |
| 4                 | be/0       | root               | 0.00 B/s   | 0.00 B/s   | 0.00 % 0.00 % [kworker/0:0H]                                        |
| 6                 | be/0       | root               | 0.00 B/s   | 0.00 B/s   | 0.00 % 0.00 % [mm_percpu_wq]                                        |
| 7                 | be/4       | root               | 0.00 B/s   | 0.00 B/s   | 0.00 % 0.00 % [ksoftirqd/0]                                         |

Figure 14.3: Using iotop

Another very useful utility is **iotop**, which must be run as root. It displays a table of current I/O usage and updates periodically. Please note that the **be** and **rt** entries in the **PRIOR** are explained in the **ionice** section, and stand for **best effort** and **real time**.

```
student@ubuntu:~$ iotop --help
Usage: /usr/sbin/iotop [OPTIONS]

DISK READ and DISK WRITE are the block I/O bandwidth used during the sampling
period. SWAPIN and IO are the percentages of time the thread spent respectively
while swapping in and waiting on I/O more generally. PRIOR is the I/O priority at
which the thread is running (set using the ionice command).

Controls: left and right arrows to change the sorting column, r to invert the
sorting order, o to toggle the --only option, p to toggle the --processes
option, a to toggle the --accumulated option, i to change I/O priority, q to
quit, any other key to force a refresh. crypto/asymmetric_keys
 CLEAN crypto
 CLEAN kernel
Options:
 --version show program's version number and exit
 -h, --help show this help message and exit
 -o, --only only show processes or threads actually doing I/O
 -b, --batch non-interactive mode
 -n NUM, --iter=NUM number of iterations before ending [infinite]
 -d SEC, --delay=SEC delay between iterations [1 second] table$ cd /tmp
 -p PID, --pid=PID processes/threads to monitor [all]
 -u USER, --user=USER users to monitor [all] gnome-screenshot -i -w
 -P, --processes only show processes, not all threads
 -a, --accumulated show accumulated I/O instead of bandwidth
 -k, --kilobytes use kilobytes instead of a human friendly unit
 -t, --time add a timestamp on each line (implies --batch)
 -q, --quiet suppress some lines of header (implies --batch) 100%
student@ubuntu:~$
```

Figure 14.4: iotop options

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 14.4 ionice \*\*

# ionice

- **ionice** sets I/O **scheduling class** and **priority**

```
$ ionice [-c class] [-n priority] [-p pid] [COMMAND [ARGS]]
```

Table 14.1: I/O Scheduling Classes

| Scheduling Class       | -c value | Meaning                                                                                                                |
|------------------------|----------|------------------------------------------------------------------------------------------------------------------------|
| <b>None or Unknown</b> | 0        | Default Value                                                                                                          |
| <b>Real Time</b>       | 1        | Get first access to the disk, can starve other processes. The priority defines how big a time slice each process gets. |
| <b>Best Effort</b>     | 2        | All programs serviced in round-robin fashion, according to priority settings. The Default.                             |
| <b>Idle</b>            | 3        | No access to disk I/O unless no other program has asked for it for a defined period.                                   |



### ionice and CFQ are no longer used

ionice works only when using the **CFQ** I/O Scheduler, which was removed in the 5.0 kernel version.

The **ionice** utility lets you set both the I/O scheduling class and priority for a given process.

The **-c** parameter specifies the I/O scheduling class as shown in the above table.

If a **pid** is given with the **-p** argument results apply to the requested process, otherwise it is the process that will be started by **COMMAND** with possible arguments. If no arguments are given, **ionice** returns the scheduling class and priority of the current shell process:

```
$ ionice
```

```
1 idle: prio 7
```

The **Best Effort** and **Real Time** classes take the **-n** argument which gives the priority, which can range from 0 to 7, with 0 being the highest priority. An example:

```
$ ionice -c 2 -n 3 -p 30078
```

Note: **ionice** works only when using the **CFQ** I/O Scheduler, which was removed in the 5.0 kernel version. We will talk about I/O schedulers in the next section.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 14.5 Labs

### Exercise 14.1: bonnie++

**bonnie++** is a widely available benchmarking program that tests and measures the performance of drives and filesystems. It is descended from **bonnie**, an earlier implementation.

Results can be read from the terminal window or directed to a file, and also to a **csv** format (**comma separated value**). Companion programs, **bon\_csv2html** and **bon\_csv2txt**, can be used convert to html and plain text output formats.

We recommend you read the **man** page for **bonnie++** before using as it has quite a few options regarding which tests to perform and how exhaustive and stressful they should be. A quick synopsis is obtained with:

```
$ bonnie++ --help
1 bonnie++: invalid option -- '-'
2 usage:
3 bonnie++ [-d scratch-dir] [-c concurrency] [-s size(MiB) [:chunk-size(b)]]
4 [-n number-to-stat[:max-size[:min-size][:num-directories[:chunk-size]]]]
5 [-m machine-name] [-r ram-size-in-MiB]
6 [-x number-of-tests] [-u uid-to-use:gid-to-use] [-g gid-to-use]
7 [-q] [-f] [-b] [-p processes | -y] [-z seed | -Z random-file]
8 [-D]
9
10 Version: 1.98
```

A quick test can be obtained with a command like:

```
$ time sudo bonnie++ -n 0 -u 0 -r 100 -f -b -d /mnt
```

where:

- **-n 0** means don't perform the file creation tests.
- **-u 0** means run as root.
- **-r 100** means pretend you have 100 MB of RAM.
- **-f** means skip per character I/O tests.
- **-b** means do a **fsync** after every write, which forces flushing to disk rather than just writing to cache.
- **-d /mnt** just specifies the directory to place the temporary file created; make sure it has enough space, in this case 300 MB, available.

If you don't supply a figure for your memory size, the program will figure out how much the system has and will create a testing file 2-3 times as large. We are not doing that here because it takes much longer to get a feel for things.



#### On RedHat

On an **RHEL/CentOS 8** system:

```
c8:/tmp>sudo bonnie++ -n 0 -u 0 -r 100 -f -b -d /mnt
```

```
1 Using uid:0, gid:0.
2 Writing intelligently...done
3 Rewriting...done
4 Reading intelligently...done
5 start 'em...done...done...done...done...
6 Version 1.98 -----Sequential Output----- --Sequential Input-- --Random-
7 -Per Chr- --Block-- -Rewrite- -Per Chr- --Block-- --Seeks--
```

For the exclusive use of LFS301 corp class taught 06 to 09 December



```

8 | Name:Size etc /sec %CP /sec %CP /sec %CP /sec %CP /sec %CP /sec %CP
9 | c8 300M 271m 20 288m 15 +++++ +++ 3916 22
10 | Latency 30us 20us 12us 18075us
11 |
12 | 1.98,1.98,c8,1,1616174245,300M,,8192,5,,,277062,20,294543,15,,,+++++,++,3916,22,,,,,,,,,\n
13 | 30us,20us,,12us,18075us,,,,,

```



## On Ubuntu

On an **Ubuntu 20.04** system, running as a virtual machine under the **VMware** hypervisor on the same physical machine:

```
$ sudo bonnie++ -n 0 -u 0 -r 100 -f -b -d /mnt
```

```

1 sing uid:0, gid:0.
2 Writing intelligently...done
3 Rewriting...done
4 Reading intelligently...done
5 start 'em...done...done...done...done...done...
6 Version 1.98 -----Sequential Output----- --Sequential Input- --Random-
7 -Per Chr- --Block-- -Rewrite- -Per Chr- --Block-- --Seeks--
8 Name:Size etc /sec %CP /sec %CP /sec %CP /sec %CP /sec %CP /sec %CP
9 ubuntu 300M 287m 88 +++++ +++ +++++ +++ +++++ ++++
10 Latency 3805us 4650us 2136us 1852us
11
12 | 1.98,1.98,ubuntu,1,1616156268,300M,,8192,5,,,293955,88,+++++,++,,,+++++,++,+++++,++,,,,,,\n
13 | ,,,3805us,4650us,,2136us,1852us,,,,,

```

You can clearly see the drop in performance. Assuming you have saved the previous outputs as a file called `bonnie++.out`, you can convert the output to html:

```
$ bon_csv2html < bonnie++.out > bonnie++.html
```

or to plain text with:

```
$ bon_csv2txt < bonnie++.out > bonnie++.txt
```

After reading the documentation, try longer and larger, more ambitious tests. Try some of the tests we turned off. If your system is behaving well, save the results for future benchmarking comparisons when the system is sick.

## Exercise 14.2: fs\_mark

The **fs\_mark** benchmark gives a low level bashing to file systems, using heavily asynchronous I/O across multiple directories and drives. It's a rather old program written by Rick Wheeler that has stood the test of time. It can be downloaded from <http://sourceforge.net/projects/fsmark/>. Once you have obtained the tarball, you can unpack it and compile it with:

```
$ tar zxvf fs_mark-3.3.tgz
$ cd fs_mark
$ make
```

Read the README file as we are only going to touch the surface. If the compile fails with an error like:

```
$ make
```

```

1 ...
2 /usr/bin/ld: cannot find -lc

```

For the exclusive use of LFS301 corp class taught 06 to 09 December

it is because you haven't installed the **static** version of **glibc**. You can do this on **Red Hat**-based systems by doing:

```
$ sudo dnf install glibc-static
```

and on **SUSE**-related systems with:

```
$ sudo zypper install glibc-devel-static
```

On **Debian**-based systems the relevant static library is installed along with the shared one so no additional package needs to be sought.



### On RedHat, Centos, or Fedora

- On **RHEL 8** `glibc-static` is available in the `codeready-builder` repo which may need to be enabled.
- However, on **CentOS 8** it is included in the **PowerTools** repo. If you already have `/etc/yum.repos.d/CentOS-PowerTools.repo` on your system, then just make sure it has `enabled=1` set in the file. Otherwise we have made a copy of this file and placed it in the **SOLUTIONS** section for this course:

For a test we are going to create 2500 files, each 10 KB in size, and after each write we'll perform an **fsync** to flush out to disk. This can be done in the `/tmp` directory with the command:

```
$ fs_mark -d /tmp -n 2500 -s 10240
```

While this is running, gather extended **iostat** statistics with:

```
$ iostat -x -d /dev/sda 2 10
```

in another terminal window.

The numbers you should surely note are the number of files per second reported by **fs\_mark** and the bandwidth percentage utilized as reported by **iostat**. If this is approaching 100 percent, you are I/O-bound.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Chapter 15

## I/O Scheduling \*\*



|      |                                 |     |
|------|---------------------------------|-----|
| 15.1 | I/O Scheduling . . . . .        | 186 |
| 15.2 | I/O Scheduler Choices . . . . . | 187 |
| 15.3 | Labs . . . . .                  | 188 |



### Please Note

\*\* These sections may be considered in part or in whole as optional. They contain either background reference material, specialized topics, or advanced subjects. The instructor may choose to cover or not cover them depending on classroom experience and time constraints.

## 15.1 I/O Scheduling

# Disk Bottlenecks and I/O Scheduling

- I/O scheduler stands between low-level device drivers and I/O requests
- Need to balance:
  - Hardware Access Time
  - Latency
  - Deadlines
  - Fairness
  - Efficiency



### Very Important

Modern SSDs can change requirements radically

The I/O scheduler provides the interface between the generic block layer and low-level physical device drivers. Both the VM (Virtual Memory) and VFS (Virtual File System) layers submit I/O requests to block devices; it is the job of the I/O scheduling layer to prioritize and order these requests before they are given to the block devices.

Any I/O scheduling algorithm has to satisfy certain (sometimes conflicting) requirements:

- Hardware access times should be minimized; i.e., requests should be ordered according to physical location on the disk. This leads to an elevator scheme where requests are inserted in the pending queue in physical order.
- Requests should be merged to the extent possible to get as big a contiguous region as possible, which also minimizes disk access time.
- Requests should be satisfied with as low a latency as is feasible; indeed, in some cases, determinism (in the sense of deadlines) may be important.
- Write operations can usually wait to migrate from caches to disk without stalling processes. Read operations, however, almost always require a process to wait for completion before proceeding further. Favoring reads over writes leads to better parallelism and system responsiveness.
- Processes should share the I/O bandwidth in a fair, or at least consciously prioritized fashion; even if it means some overall performance slowdown of the I/O layer, process throughput should not suffer inordinately.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 15.2 I/O Scheduler Choices

### I/O Scheduler Choices

- To see which I/O schedulers are available (for disk `/dev/sda`):

```
$ cat /sys/block/sda/queue/scheduler
noop [deadline] cfq # Before kernel version 5.0
none bfq kyber [mq-deadline] # After kernel version 5.0
```

- To change the value (as superuser):

```
echo bfq > /sys/block/sda/queue/scheduler
$ cat /sys/block/sda/queue/scheduler
none [bfq] kyber mq-deadline
```

- Scheduler-specific tunables can be found in `/sys/block/sda/queue/iosched`

Since demands can be conflicting, different I/O schedulers may be appropriate for different workloads; e.g., a large database server vs. a desktop system. In order to provide flexibility, the **Linux** kernel has an object oriented scheme, in which pointers to the various needed functions are supplied in a data structure, the particular one of which can be selected at run time.

At least one of the I/O scheduling algorithms must be compiled into the kernel. Exactly what is available has changed with time, with a major overhaul coming with the 5.0 **Linux** kernel version.

The default choice is a compile configuration option; for kernels before 2.6.18 it was **AS**, it then became **CFQ** and is now **deadline-mq**, but distributions may have a different preference.

It is possible to use different I/O schedulers for different devices.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 15.3 Labs

### Exercise 15.1: Comparing I/O Schedulers

We provide a script which is to be used to compare I/O schedulers which can be extracted from your downloaded SOLUTIONS file as `lab_iosched.sh`.

```
sh lab_iosched.sh

#!/bin/bash

NMAX=8
NMEGS=100
[[-n $1]] && NMAX=$1
[[-n $2]] && NMEGS=$2

echo Doing: $NMAX parallel read/writes on: $NMEGS MB size files

TIMEFORMAT="%R %U %S"

#####
simple test of parallel reads
do_read_test(){
 for n in $(seq 1 $NMAX) ; do
 cat file$n > /dev/null &
 done
 # wait for previous jobs to finish
 wait
}

simple test of parallel writes
do_write_test(){
 for n in $(seq 1 $NMAX) ; do
 [[-f fileout$n]] && rm -f fileout$n
 (cp file1 fileout$n && sync) &
 done
 # wait for previous jobs to finish
 wait
}

create some files for reading, ok if they are the same
create_input_files(){
 [[-f file1]] || dd if=/dev/urandom of=file1 bs=1M count=$NMEGS
 for n in $(seq 1 $NMAX) ; do
 [[-f file$n]] || cp file1 file$n
 done
}

echo -e "\ncreating as needed random input files"
create_input_files

#####
begin the actual work

do parallel read test
echo -e "\ndoing timings of parallel reads\n"
echo -e " REAL USER SYS\n"
#for iosched in noop deadline cfq ; do
for iosched in \
 $(cat /sys/block/sda/queue/scheduler | sed -e s/'\['//g -e s/'\]'//g) ; do
```

For the exclusive use of LFS301 corp class taught 06 to 09 December



```

echo testing IOSCHED = $iosched
echo $iosched > /sys/block/sda/queue/scheduler
cat /sys/block/sda/queue/scheduler
echo -e "\nClearing the memory caches\n"
echo 3 > /proc/sys/vm/drop_caches
time do_read_test
done
#####
do parallel write test
echo -e "\ndoing timings of parallel writes\n"
echo -e " REAL USER SYS\n"
for iosched in \
$(cat /sys/block/sda/queue/scheduler | sed -e s/'\\[\\]'//g -e s/'\\]'//g) ; do
echo testing IOSCHED = $iosched
echo $iosched > /sys/block/sda/queue/scheduler
cat /sys/block/sda/queue/scheduler
time do_write_test
done
#####

```

If you are taking the online self-paced version of this course, the script is available for download from your **Lab** screen.

Because changing the I/O scheduler is a privileged operation, you will have to run it as:

```
$ sudo ./lab_iosched.sh [# reads/writes (NMAX)] [file size in MB (NMEGS)]
```



## How it works

The script:

- Cycles through the available I/O schedulers on a hard disk while doing a configurable number of parallel reads and writes of files of a configurable size. (Note that exactly which schedulers are available will depend on the kernel has been configured and compiled and may vary quite a bit from machine to machine.)
- Tests reads and writes as separate steps.
- Makes sure, when testing reads, it is actually reading from disk and not from cached pages of memory; the cache is flushed out by doing (as root):
 

```
echo 3 > /proc/sys/vm/drop_caches
```

 before doing the reads. The script does a **cat** into `/dev/null` to avoid writing to disk.
- Makes sure all reads are complete before obtaining timing information; this is done by issuing a **wait** command under the shell.
- Tests writes by simply copying a file (which will be in cached memory after the first read) multiple times simultaneously. To make sure it has waited for all writes to complete before getting timing information, it issues a **sync** call.

The provided script takes two arguments. The first is the number of simultaneous reads and writes to perform. The second is the size (in MB) of each file.

This script must be run as root as it echoes values into the `/proc` and `/sys` directory trees.

Compare the results you obtain using different I/O schedulers.

For the exclusive use of LFS301 corp class taught 06 to 09 December

**Extra Credit**

For additional exploring you might try changing some of the tunable parameters and see how results vary.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 16

# Linux Filesystems and the VFS



|      |                                    |     |
|------|------------------------------------|-----|
| 16.1 | Filesystem Basics . . . . .        | 192 |
| 16.2 | Filesystem Concepts . . . . .      | 193 |
| 16.3 | Virtual Filesystem (VFS) . . . . . | 195 |
| 16.4 | Available Filesystems . . . . .    | 196 |
| 16.5 | Journalling Filesystems . . . . .  | 198 |
| 16.6 | Special Filesystems . . . . .      | 199 |
| 16.7 | Labs . . . . .                     | 200 |

## 16.1 Filesystem Basics

# Filesystem Basics

- **Linux** programs read and write files, not disk sectors
- A **file** is actually an abstraction camouflaging the physical I/O layer
- **Filesystems** create a usable format on a physical partition
- A **UNIX**-like filesystem uses a tree hierarchy
  - Directories contain files and/or other directories
  - Every path or node is under the root (`/`) directory
- Multiple filesystems may be (and usually are) merged together into a single tree structure
- **Linux** uses a virtual filesystem layer (**VFS**) to communicate with the filesystem software

Application programs read and write **files**, rather than dealing with physical locations on the actual hardware on which files are stored.

Files and their names are an abstraction camouflaging the physical I/O layer. Directly writing to disk from the command line (ignoring the **filesystem** layer) is very dangerous and is usually only done by low-level operating system software and not by user applications. An exception is some very high-end software such as enterprise data bases that do such **cmd** access to skip filesystem-related latency.

Local filesystems generally reside within a disk partition which can be a physical partition on a disk, or a logical partition controlled by a **Logical Volume Manager (LVM)**. Filesystems can also be of a network nature and their true physical embodiment completely hidden to the local system across the network.

## 16.2 Filesystem Concepts

### Inodes

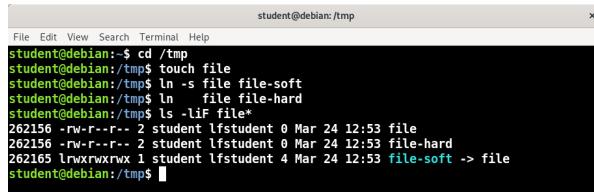
- The **name** of a file is just a property of its **inode**, which is the more fundamental object.
- Inodes are stored on the filesystem as well as being data structures in memory.
- Inodes describe and store information about a file including:
  - Permissions
  - User, group ownership
  - Size
  - Timestamps (nanosecond)
    - \* Last access time
    - \* Last modification time
    - \* Change time
- Filenames are **not** stored in the inode; they are stored in the **directory**

An inode is a data structure on disk that describes and stores a file's attributes, including its location.

Every file which is contained in a **Linux** filesystem is associated with its own inode. All data about a file is contained within its inode. The inode is used by the operating system to keep track of properties such as name, location, file attributes (permissions, ownership, etc.), access times and other items. Because of this, all I/O activity concerning a file usually also involves the file's inode.

# Hard and Soft Links

- **Hard** links point to an inode.
  - Two or more files can point to the same inode (hard link)
  - All hard linked files have to be on the same filesystem
- **Soft** (or **symbolic**) links point to a file name with an associated inode.
  - Soft linked files may be on different filesystems
  - The target may not exist or yet be mounted, it can be **dangling**
- Be careful with hard linked files:
  - Changing the contents in one place may not change it at others



```
student@debian:/tmp$ cd /tmp
student@debian:/tmp$ touch file
student@debian:/tmp$ ln -s file file-soft
student@debian:/tmp$ ln file file-hard
student@debian:/tmp$ ls -lif file*
262156 -rw-r--r-- 2 student lfstudent 0 Mar 24 12:53 file
262156 -rw-r--r-- 2 student lfstudent 0 Mar 24 12:53 file-hard
262165 lrwxrwxrwx 1 student lfstudent 4 Mar 24 12:53 file-soft --> file
student@debian:/tmp$
```

Figure 16.1: Hard and Soft Links

A **directory file** is a particular type of file that is used to associate file names and inodes. There are two ways to associate (or **link**) a file name with an inode:

- **Hard** links point to an inode. They are made by using **ln** without an option.
- **Soft** (or **symbolic**) links point to a file name which has an associated inode. They are made with using **ln** with the **-s** option.

Each association of a directory file contents and an inode is known as a link. Additional links can be created using **ln**.

Because it is possible (and quite common) for two or more directory entries to point to the same inode (hard links), a file can be known by multiple names, each of which has its own place in the directory structure. However, it can have only one inode no matter which name is being used.

When a process refers to a pathname, the kernel searches directories to find the corresponding inode number. After the name has been converted to an inode number, the inode is loaded into memory and is used by subsequent requests.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 16.3 Virtual Filesystem (VFS)

# VFS

The **Virtual FileSystem (VFS)**:

- Abstraction layer
- Communicates between the kernel and the real filesystems (on all storage media)
- Real filesystems register with the **VFS** layer

**Linux** implements a **Virtual File System (VFS)**, as do all modern operating systems. When an application needs to access a file it interacts with the **VFS** abstraction layer, which then translates all the I/O system calls (reading, writing etc.) into specific code relevant to the particular actual filesystem.

Thus neither the specific actual filesystem or physical media and hardware on which it resides need be considered by applications. Furthermore, network filesystems (such as **NFS**) can be handled transparently.

This permits **Linux** to work with more filesystem varieties than any other operating system. This democratic attribute has been a large factor in its success.

Most filesystems have full read and write access while a few have only read access and perhaps experimental write access. Some filesystem types, especially non-**UNIX** based ones, may require more manipulation in order to be represented in the **VFS**.

Variants such as **vfat** do not have distinct read/write/execute permissions for the owner/group/world fields; the **VFS** has to make an assumption about how to specify distinct permissions for the three types of user, and such behavior can be influenced by mounting operations. There are non-kernel filesystem implementations, such as the read/write **ntfs-3g** (<https://sourceforge.net/projects/ntfs-3g/>) which are reliable but have weaker performance than in-kernel filesystems.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 16.4 Available Filesystems

# Available Filesystems

- **Linux** works with more filesystem varieties than any other operating system.
- This democratic flexibility has been a large factor in its success.
- Most filesystems have full read/write access, while a few have read only access.
- Commonly used filesystems include **ext4**, **xfs**, **btrfs**, **squashfs**, **nfs** and **vfat**
- A list of currently supported filesystems can be seen at </proc/filesystems>

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Filesystem Varieties

**Linux** supports many filesystem varieties, most with full read and write access, including:

- **ext4**: **Linux** native filesystem (and earlier **ext2** and **ext3**)
- **XFS**: A high-performance filesystem originally created by **SGI**
- **JFS**: A high-performance filesystem originally created by **IBM**
- **Windows**-natives: **FAT12**, **FAT16**, **FAT32**, **VFAT**, **NTFS**
- Pseudo-filesystems resident only in memory, including **proc**, **sysfs**, **debugfs**
- Network filesystems such as **NFS**, **coda**, **afs**
- etc.

You can see a list of the filesystem types currently registered and understood by the currently running **Linux** kernel by doing:

```
$ cat /proc/filesystems
```

```
1 iso9660
2 squashfs
3 ext3
4 ext2
5 ext4
6 fuseblk
7 nodev sysfs
8 nodev proc
9 nodev tmpfs
10 nodev debugfs
11 nodev sockfs
12 nodev hugetlbfs
13 nodev fuse
14 nodev nfsd
15
```

(The ones with `nodev` are **special filesystems** which do not reside on storage.) Additional filesystems may have their code loaded as a module only when the system tries to access a partition that uses them.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 16.5 Journalling Filesystems

# Journalling Filesystems

- Recover gracefully from crashes or abnormal shutdown
- Do recovery and repair extremely quickly
- Work with **atomic transactions** which must be fully completed or are revoked.
- Work with **journal log files**
- Include many advanced features
- Currently available: **ext4**, **ext3**, **xfs**, **reiserfs**, and **btrfs**

**Journalling** filesystems recover from system crashes or ungraceful shutdowns with little or no corruption, and they do so very rapidly. While this comes at the price of having some more operations to do, additional enhancements can more than offset the price.

In a journalling filesystem operations are grouped into **transactions**. A transaction must be completed without error, **atomically**; otherwise the filesystem is not changed. A log file is maintained of transactions. When an error occurs usually only the last transaction needs to be examined.

The following journalling filesystems are freely available under **Linux**:

- **ext3** was an extension of the earlier non-journalling **ext2** filesystem.
- **ext4** is a vastly enhanced outgrowth of **ext3**. Features include extents, 48-bit block numbers, and up to 16 TB size. Most **Linux** distributions have used **ext4** as the default filesystem for quite a few years.
- **reiserfs** was the first journalling implementation used in **Linux**, but lost its leadership and further development was abandoned.
- **JFS** was originally a product of **IBM** and was ported from **IBM's AIX** operating system.
- **XFS** was originally a product of **SGI** and was ported from **SGI's IRIX** operating systems. **RHEL 7** adopted **XFS** as its default filesystem.
- **btrfs** is the newest of the journalling filesystems and is still under rapid development. It is the default for **SUSE** and **openSUSE** systems.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 16.6 Special Filesystems

# Special Filesystems

- Special filesystems are used to gain access to or tune system internals or to implement certain functions.
- Some have mount points (such as **proc** at `/proc` or **sys** at `/sys`)
- Some have no mount points (such as **sockfs** or **pipefs**)
- These special filesystems are really not true filesystems; they are kernel facilities or subsystems that find the filesystem structural abstraction to be a useful way to organize data and functionality.

**Linux** widely employs the use of **special filesystems** for certain tasks. These are particularly useful for accessing various kernel data structures and tuning kernel behaviour, or for implementing particular functions. A partial list:

Table 16.1: Special Filesystems

| Filesystem         | Mount Point                    | Purpose                                                             |
|--------------------|--------------------------------|---------------------------------------------------------------------|
| <b>rootfs</b>      | None                           | During kernel load, provides an empty root directory.               |
| <b>hugetlbfs</b>   | Anywhere                       | Provides extended page access (2 or 4 MB on <b>x86</b> )            |
| <b>bdev</b>        | None                           | Used for block devices.                                             |
| <b>proc</b>        | <code>/proc</code>             | Pseudo filesystem access to many kernel structures and sub-systems. |
| <b>sockfs</b>      | None                           | Used by <b>BSD Sockets</b>                                          |
| <b>tmpfs</b>       | Anywhere                       | RAM disk with swapping, resizing.                                   |
| <b>shm</b>         | None                           | Used by System V IPC Shared Memory.                                 |
| <b>pipefs</b>      | None                           | Used for pipes.                                                     |
| <b>binfmt_misc</b> | Anywhere                       | Used by various executable formats.                                 |
| <b>devpts</b>      | <code>/dev/pts</code>          | Used by <b>Unix98</b> pseudo-terminals                              |
| <b>usbfs</b>       | <code>/proc/bus/usb</code>     | Used by <b>USB</b> sub-system for dynamical devices.                |
| <b>sysfs</b>       | <code>/sys</code>              | Used as a <b>device tree</b> .                                      |
| <b>debugfs</b>     | <code>/sys/kernel/debug</code> | Used for simple debugging file access.                              |

Note that some of these special filesystems have no mount point; this means user applications do not interact with them, but the kernel uses them, taking advantage of VFS layers and code.

FOR the exclusive use of LFS301 corp class taught 06 to 09 December

## 16.7 Labs



### Video Demonstration Resources

[using\\_available\\_filesystems\\_demo.mp4](#)

## Exercise 16.1: The tmpfs Special Filesystem

**tmpfs** is one of many special filesystems used under **Linux**. Some of these are not really used as filesystems, but just take advantage of the filesystem abstraction. However, **tmpfs** is a real filesystem that applications can do I/O on.

Essentially, **tmpfs** functions as a **ramdisk**; it resides purely in memory. But it has some nice properties that old-fashioned conventional ramdisk implementations did not have:

1. The filesystem adjusts its size (and thus the memory that is used) dynamically; it starts at zero and expands as necessary up to the maximum size it was mounted with.
2. If your RAM gets exhausted, **tmpfs** can utilize swap space. (You still can't try to put more in the filesystem than its maximum capacity allows, however.)
3. **tmpfs** does not require having a normal filesystem placed in it, such as **ext3** or **vfat**; it has its own methods for dealing with files and I/O that are aware that it is really just space in memory (it is not actually a block device), and as such are optimized for speed.

Thus there is no need to pre-format the filesystem with a **mkfs** command; you merely just have to mount it and use it.

Mount a new instance of **tmpfs** anywhere on your directory structure with a command like:

```
$ sudo mkdir /mnt/tmpfs
$ sudo mount -t tmpfs none /mnt/tmpfs
```

See how much space the filesystem has been given and how much it is using:

```
$ df -h /mnt/tmpfs
```

You should see it has been allotted a default value of half of your RAM; however, the usage is zero, and will only start to grow as you place files on **/mnt/tmpfs**.

You could change the allotted size as a mount option as in:

```
$ sudo mount -t tmpfs -o size=1G none /mnt/tmpfs
```

You might try filling it up until you reach full capacity and see what happens. Do not forget to unmount when you are done with:

```
$ sudo umount /mnt/tmpfs
```

Virtually all modern **Linux** distributions mount an instance of **tmpfs** at **/dev/shm**:

```
$ df -h /dev/shm
```

|   |            |       |      |      |       |      |            |
|---|------------|-------|------|------|-------|------|------------|
| 1 | Filesystem | Type  | Size | Used | Avail | Use% | Mounted on |
| 2 | tmpfs      | tmpfs | 3.9G | 24M  | 3.9G  | 1%   | /dev/shm   |

Many applications use this such as when they are using **POSIX** shared memory as an inter-process communication mechanism. Any user can create, read and write files in **/dev/shm**, so it is a good place to create temporary files in memory.

Create some files in **/dev/shm** and note how the filesystem is filling up with **df**.

In addition, many distributions mount multiple instances of **tmpfs**; for example, on a **RHEL** system:

For the exclusive use of LFS301 corp class taught 06 to 09 December

```
$ df -h | grep ' tmpfs'
```

|   |       |       |      |      |      |    |                |
|---|-------|-------|------|------|------|----|----------------|
| 1 | tmpfs | tmpfs | 7.8G | 38M  | 7.8G | 1% | /dev/shm       |
| 2 | tmpfs | tmpfs | 7.8G | 18M  | 7.8G | 1% | /run           |
| 3 | tmpfs | tmpfs | 7.8G | 0    | 7.8G | 0% | /sys/fs/cgroup |
| 4 | tmpfs | tmpfs | 1.6G | 1.2M | 1.6G | 1% | /run/user/42   |
| 5 | tmpfs | tmpfs | 1.6G | 56K  | 1.6G | 1% | /run/user/1000 |

Notice this was run on a system with 16 GB of ram, so clearly you cannot have all these **tmpfs** filesystems actually using the default ~8 GB they have each been allotted!



### Please Note

Some distributions (such as **Fedora**) may (by default) mount `/tmp` as a **tmpfs** system; in such cases one has to avoid putting large files in `/tmp` to avoid running out of memory. Or one can disable this behavior as we discussed earlier when describing `/tmp`.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 17

# Disk Partitioning



|       |                                                     |     |
|-------|-----------------------------------------------------|-----|
| 17.1  | Common Disk Types . . . . .                         | 204 |
| 17.2  | Disk Geometry . . . . .                             | 205 |
| 17.3  | Partitioning . . . . .                              | 206 |
| 17.4  | Partition Tables . . . . .                          | 208 |
| 17.5  | Naming Disk Devices . . . . .                       | 210 |
| 17.6  | blkid and lsblk . . . . .                           | 211 |
| 17.7  | Sizing up partitions . . . . .                      | 213 |
| 17.8  | Backing Up and Restoring Partition Tables . . . . . | 214 |
| 17.9  | Partition table editors . . . . .                   | 215 |
| 17.10 | fdisk . . . . .                                     | 216 |
| 17.11 | Labs . . . . .                                      | 217 |

## 17.1 Common Disk Types

# Common Disk Types

- **SATA** (Serial AT Attachment)
  - Faster data transfer, smaller cable than **IDE**
  - Seen as **SCSI** devices
- **SCSI** (Small Computer Systems Interface)
  - Generally faster
  - Numerous versions such as Fast, Wide, and Ultra, Ultrawide
- **SAS** (Serial Attached SCSI)
  - Newer point to point serial protocol
  - Better performance than **SATA**
- **USB**
  - Includes flash drives and floppies
  - Seen as SCSI devices
- **SSD**
  - No moving parts, attach to regular controllers
- **IDE** and **EIDE** (Integrated Drive Electronics, Enhanced IDE)
  - Obsolete

There are a number of different hard disk types, each of which is characterized by the type of **data bus** they are attached to, and other factors such as speed, capacity, how well multiple drives work simultaneously etc.

**SATA** disks were designed to replace the old **IDE** drives. They offer a smaller cable size (7 pins), native hot swapping, and faster and more efficient data transfer.

**SAS** is better suited for servers. See <https://store.hp.com/us/en/tech-takes/sas-vs-sata> for a clear and simple explanation of the differences.

**SCSI** disks range from narrow (8 bit bus) to wide (16 bit bus) with a transfer rate of from about 5 MB per second (narrow, standard **SCSI**) to about 160 MB per second (**Ultra-Wide SCSI-3**).

Modern **SSD** drives (Solid State Drive) have come down in price, have no moving parts, and use less power than drives with rotational media, and have faster transfer speeds. Internal **SSDs** are even installed with the same form factor and in the same enclosures as conventional drives.

**SSDs** still cost quite a bit more but price is decreasing. It is common to have both **SSDs** and rotational drives in the same machines, with frequently accessed and performance critical data transfers taking place on the **SSDs**.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 17.2 Disk Geometry

# Disk Geometry

- **Rotational** disks are composed of one or more platters and each platter is read by one or more heads
- Heads read a circular track off a platter as the disk spins
- Circular tracks are divided into data blocks called sectors
- Originally, sectors were 512 bytes of data; now usually 4096 bytes
- A cylinder is a group which consists of the same track on all platters
- For **SSDs** these geometry concepts make no sense

```
$ sudo fdisk -l /dev/sdc |grep -i sector
```

```
Disk /dev/sdc: 1.8 TiB, 2000398934016 bytes, 3907029168 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
```

The physical structural image has become less and less relevant as internal electronics on the drive actually obscure much of it. Furthermore, **SSDs** have no moving parts or anything like the above ingredients.

We can see the geometry with **fdisk**:

```
File Edit View Search Terminal Help
c7:/tmp>sudo fdisk -l /dev/sda

Disk /dev/sda: 2000.4 GB, 2000398934016 bytes, 3907029168 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disk label type: dos
Disk identifier: 0x000852df

 Device Boot Start End Blocks Id System
/dev/sda1 2048 1048578047 524288000 8e Linux LVM
/dev/sda2 1048578048 2097154047 524288000 8e Linux LVM
/dev/sda3 2097154048 3907028991 904937472 5 Extended
/dev/sda5 2097156096 3145732095 524288000 8e Linux LVM
/dev/sda6 3890448384 3907028991 8290304 82 Linux swap / Solaris
c7:/tmp>
```

Figure 17.1: Using **fdisk**

Note the use of the **-l** option which simply lists the partition table without entering interactive mode.

Historically disks were manufactured with sectors of 512 bytes; 4 KB is now most common by far; larger sector sizes can lead to faster transfer speeds. Linux still uses a **logical** sector size of 512 bytes for backward compatibility, but this is simply for pretend in software.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 17.3 Partitioning

# Partition Organization

- Disks are divided into **partitions**
- A partition is a physically contiguous region on the disk
- On the most common architectures, there are two partitioning schemes in use:
  - **MBR** (Master Boot Record)
  - **GPT** (GUID Partition Table)
- **MBR** dates back to early days of **MSDOS**  
May have up to 4 **primary** partitions and one of the primary partitions can be designated as an **extended** partition subdivided into **logical** partitions with 15 partitions possible.
- **GPT** is on all modern systems and is based on **UEFI** (Unified Extensible Firmware Interface)  
By default may have up to 128 **primary** partitions

- When using the **MBR** scheme:

If we have a **SCSI** drive, for example `/dev/sda`, then `/dev/sda1` is the first primary partition and `/dev/sda2` is the second primary partition. If we created an extended partition `/dev/sda3` it could be divided into logical partitions. All partitions greater than four are logical partitions (meaning contained within an extended partition). There can only be one extended partition, but it can be divided into numerous logical partitions.

**Note:** **Linux** does not require partitions to begin or end on cylinder boundaries, but other operating systems might complain if they do not. For this reason, the widely deployed **Linux** partitioning utilities try to play nice and end on boundaries. Obviously partitions should not overlap either.

- When using the **GPT** scheme:

There is no need for extended partitions, and you can have (by default) 128 primary partitions.

Partitions can be up to  $2^{33}$  TB in size! (with (MBR) the limit is just 2 TB.)

## Why Partition?

- **Separation** of user and application data from operating system files
- **Sharing** between operating systems and/or machines
- **Security** enhancement by imposing different quotas and permissions for different system parts
- **Size** concerns; keeping variable and volatile storage isolated from stable
- **Performance** enhancement of putting most frequently used data on faster storage media
- **Swap** space can be isolated from data and also used for hibernation storage

Deciding what to partition and how to separate your partitions is cause for thought. The reasons to have distinct partitions include increased granularity of security, quota, settings, or size restrictions. You could also have distinct partitions to allow for data protection.

A common partition layout contains a `/boot` partition, a partition for the root filesystem `/`, a swap partition, and a partition for the `/home` directory tree.

Keep in mind that it is more difficult to re-size a partition after the fact than during install/creation time. Plan accordingly.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 17.4 Partition Tables

### MBR Partition Table

- The Partition table is in the last 64 bytes of the first sector, the 512 bytes of the **MBR** (Master Boot Record)
- Each entry in the partition table is 16 bytes long and describes one of the four possible primary partitions. The information for each is:
  - Active bit
  - Beginning address in cylinder-/head/sectors (**CHS**) format
  - Partition type code, indicating: **Linux**, **Linux LVM**, **ntfs**, **swap**, etc
  - Ending address in **CHS**
  - Start sector, counting linearly from 0
  - Number of sectors in partition

**Linux** only uses the last two fields for addressing using the linear block addressing (**LBA**) method.

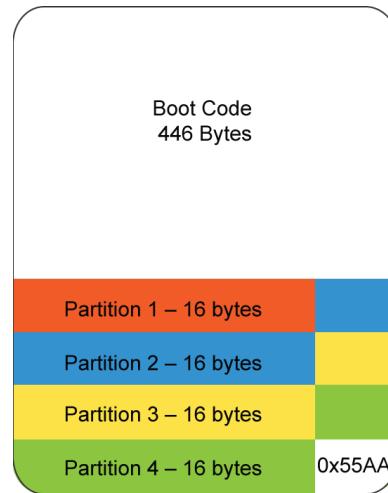


Figure 17.2: MBR Disk Partition Table

The disk partition table is contained within the disk's **MBR** and is the 64 bytes following the 446 byte boot record.

One partition on a disk may be marked active. When the system boots, that partition is where the master boot loader looks for items to load.

Remember that there can be only one extended partition, but that partition may contain a number of logical partitions.

The structure of the **MBR** is defined by an operating system-independent convention. The first 446 bytes are reserved for the program code. They typically hold part of a boot loader program. The next 64 bytes provide space for a partition table of up to four entries. The operating system needs this table for handling the hard disk.

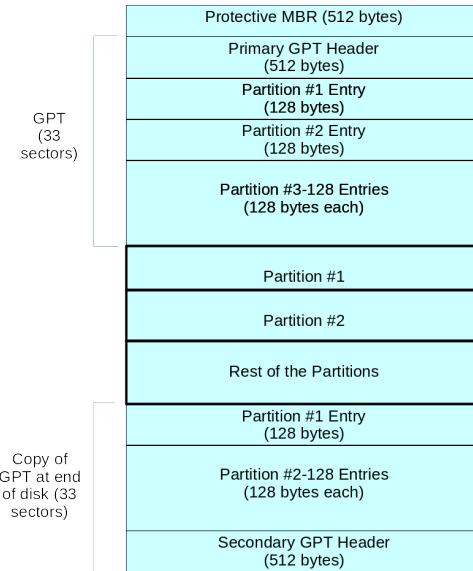
On **Linux** systems, the beginning and ending address in **CHS** is ignored.

(Note for the curious, there are 2 more bytes at the end of the **MBR** known as the magic number, signature word, or end of sector marker, which always have the value 0x55AA.)

For the exclusive use of LFS301 corp class taught 06 to 09 December

## GPT Partition Table

- Modern hardware comes with **GPT** support; **MBR** support will gradually fade away
- The Protective MBR is for backwards compatibility so UEFI systems can be booted the old way.
- Two copies of **GPT** header, at beginning and end of disk, describing metadata:
  - List usable blocks on disk
  - Number of partitions
  - Size of partition entries
- Each partition entry has a minimum size of 128 bytes

Figure 17.3: **GPT Disk Partition Table**

The **blkid** utility (to be discussed later) shows information about partitions:

- On a modern UEFI/GPT system::

```
$ sudo blkid /dev/sda8
```

```
/dev/sda8: LABEL="RHEL8" UUID="53ea9807-fd58-4433-9460-d03ec36f73a3" BLOCK_SIZE="4096" TYPE="ext4"
↪ PARTUUID="0c79e35b-e58b-4ce3-bd34-45651b01cf43"
```

- On a legacy MBR system:

```
$ sudo blkid /dev/sdb2
```

```
/dev/sdb2: LABEL="RHEL8" UUID="6921b738-1e36-429a-89be-8b97cf2f0556" BLOCK_SIZE="4096" TYPE="ext4"
↪ PARTUUID="00022650-02"
```

Note both examples give a unique UUID which describes the filesystem on the partition, not the partition itself. It changes if the filesystem is reformatted

The **GPT** partition also gives a PARTUUID which describes the partition and stays the same even if the filesystem is reformatted. If the hardware supports it is possible to migrate a **MBR** system to **GPT**, but it is not hard to brick the machine while doing so. Thus, usually the benefits are not worth the risk.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 17.5 Naming Disk Devices

# Naming Disk Devices and Device Nodes

- Device files found in `/dev` directory
- Name format: `xx[yz]`
  - `xx` is the device type (usually `sd`)
  - `y` is a letter for the drive number (`a, b, c, etc`)
  - `z` is the partition number
- Sample:
  - `/dev/sda` is the first drive found
  - `/dev/sdb2` is the second partition on the second drive

The **Linux** kernel interacts at a low level with disks through **device nodes** normally found in the `/dev` directory. Normally device nodes are accessed only through the infrastructure of the kernel's **Virtual File System**; raw access through the device nodes is an extremely efficient way to destroy a filesystem.

For an example of proper raw access, you can format a partition, as in:

```
$ sudo mkfs.ext4 /dev/sda9
```

Device nodes for **SCSI** and **SATA** disks follow a simple naming convention:

- The first hard disk is `/dev/sda`.
- The second hard disk is `/dev/sdb`.
- etc.

Partitions are also easily enumerated as in:

- `/dev/sdb1` is the first partition on the second disk.
- `/dev/sdc4` is the fourth partition on the third disk.

In the above `sd` means **SCSI** or **SATA** disk. Back in the days where **IDE** disks could be found they would have been `/dev/hda3`, `/dev/hdb` etc.

Doing `ls -l /dev` will show you the current available disk device nodes.

**For the exclusive use of LFS301 corp class taught 06 to 09 December**

## 17.6 blkid and lsblk

# blkid

- Command line utility to locate and print block device attributes
- Report on content of block device
- Report on attributes of block device
- Sample commands:

```
$ sudo blkid
$ sudo blkid /dev/sda*
$ sudo blkid -L root
```

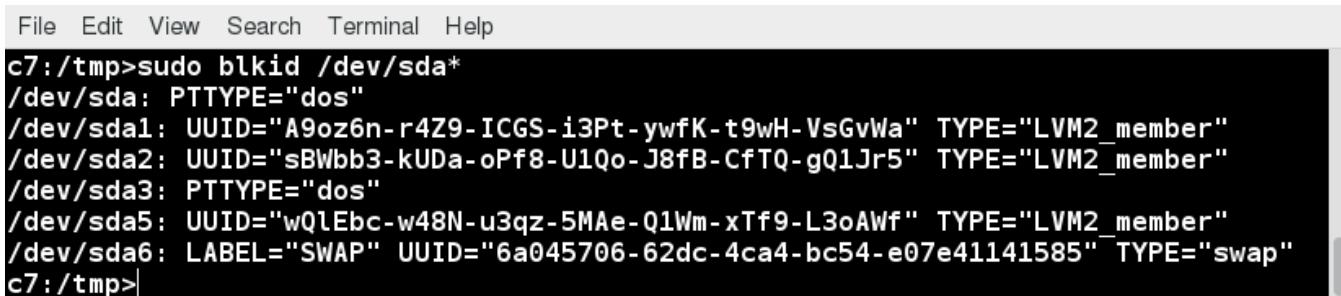
**blkid** is a utility to locate block devices and report on their attributes. It works with the **libblkid** library. It can take as an argument a particular device or list of devices.

It can determine the type of content (e.g. filesystem, swap) a block device holds, and also attributes (tokens, NAME=value pairs) from the content meta-data (e.g. LABEL or UUID fields).

**blkid** will only work on devices which contain data that is finger-printable; e.g., an empty partition will not generate a block-identifying **UUID**.

**blkid** has two main forms of operation: either searching for a device with a specific NAME=value pair, or displaying NAME=value pairs for one or more devices.

Without arguments it will report on all devices. There are quite a few options designating how to specify devices and what attributes to report on.



```
File Edit View Search Terminal Help
c7:/tmp>sudo blkid /dev/sda*
/dev/sda: PTTYPE="dos"
/dev/sda1: UUID="A9oz6n-r4Z9-ICGS-i3Pt-ywfK-t9wH-VsGvWa" TYPE="LVM2_member"
/dev/sda2: UUID="sBWbb3-kUDa-oPf8-U1Qo-J8fB-CfTQ-gQ1Jr5" TYPE="LVM2_member"
/dev/sda3: PTTYPE="dos"
/dev/sda5: UUID="wQLEbc-w48N-u3qz-5MAe-Q1Wm-xTf9-L3oAWf" TYPE="LVM2_member"
/dev/sda6: LABEL="SWAP" UUID="6a045706-62dc-4ca4-bc54-e07e41141585" TYPE="swap"
c7:/tmp>
```

Figure 17.4: Using **blkid**

For the exclusive use of LFS301 corp class taught 06 to 09 December

# lsblk

- Presents block device information in tree format

```
File Edit View Search Terminal Help
c7:/tmp>lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sdb 8:16 0 238.5G 0 disk
└─sdb2 8:18 0 1K 0 part
 └─sdb7 8:23 0 27G 0 part
 └─VG-vms 253:12 0 184G 0 lvm /VMS
 └─sdb5 8:21 0 128G 0 part
 └─VG-local 253:10 0 24G 0 lvm /usr/local
 └─VG-src 253:11 0 11G 0 lvm /usr/src
 └─VG-vms 253:12 0 184G 0 lvm /VMS
 └─sdb1 8:17 0 19.5G 0 part /
 └─sdb6 8:22 0 64G 0 part
 └─VG-vms 253:12 0 184G 0 lvm /VMS
loop0 7:0 0 2.5G 0 loop /usr/src/KERNELS
sda 8:0 0 1.8T 0 disk
└─sda2 8:2 0 500G 0 part
 └─VG2-P 253:6 0 125G 0 lvm
 └─VG2-virtual 253:4 0 350G 0 lvm /VIRTUAL
 └─VG2-isabelle 253:7 0 128G 0 lvm
 └─sda5 8:5 0 500G 0 part
 └─VG2-PLAY 253:9 0 100G 0 lvm
 └─sda3 8:3 0 1K 0 part
 └─sda1 8:1 0 500G 0 part
 └─VG2-dead 253:1 0 70G 0 lvm /DEAD
 └─VG2-dead2 253:8 0 100G 0 lvm /DEAD2
 └─VG2-P 253:6 0 125G 0 lvm
 └─VG2-virtual 253:4 0 350G 0 lvm /VIRTUAL
 └─VG2-iso_images 253:2 0 110G 0 lvm /ISO IMAGES
 └─VG2-pictures 253:0 0 20G 0 lvm /PICTURES
 └─VG2-w7back 253:5 0 50G 0 lvm
 └─VG2-audio 253:3 0 16G 0 lvm /AUDIO
 └─sda6 8:6 0 7.9G 0 part [SWAP]
c7:/tmp>
```

Figure 17.5: Using lsblk

```
$ lsblk -h
```

```
1 Usage:
2 lsblk [options] [<device> ...]
3
4 List information about block devices.
5
6 Options:
7 -a, --all print all devices
8 -b, --bytes print SIZE in bytes rather than in human readable format
9 -d, --nodeps don't print slaves or holders
10 -D, --discard print discard capabilities
11 -z, --zoned print zone model
12 -e, --exclude <list> exclude devices by major number (default: RAM disks)
13 -f, --fs output info about filesystems
14 -i, --ascii use ascii characters only
15 -I, --include <list> show only devices with specified major numbers
16 -J, --json use JSON output format
17 -l, --list use list format output
18 -T, --tree use tree format output
19 -m, --perms output info about permissions
20 -n, --noheadings don't print headings
21 -o, --output <list> output columns
22 -O, --output-all output all columns
23 -p, --paths print complete device path
24 -P, --pairs use key="value" output format
25 -r, --raw use raw output format
26 -s, --inverse inverse dependencies
27 -S, --scsi output info about SCSI devices
28 -t, --topology output info about topology
29 -x, --sort <column> sort output by <column>
30
31 -h, --help display this help
32 -V, --version display version
33
34 Available output columns:
35
36 For more details see lsblk(8).
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 17.7 Sizing up partitions

# Sizing up Partitions

- Most **Linux** systems should use a minimum of two partitions:  
**root** (/)
  - Used for the filesystem
  - Most **Linux** installations will have more than one filesystem
  - It is somewhat inconvenient to re-size (can use **LVM** when possible)
- Swap**
  - Used as an extension of physical memory
  - A common choice is equal to **RAM** size
  - May have multiple swap partitions as well as swap files
- Some distributions, such as **Ubuntu**, default to a swap file rather than a partition, but:
  - More flexible
  - More dangerous if error or bug

Most installations will have more than one filesystem on more than one partition, which are joined together at **mount points**.

It is difficult with most filesystem types to resize the root partition; using **LVM** (discussed later) can make this easier.

While it is certainly possible to run **Linux** with just the root partition, most systems use more partitions to allow for easier backups, more efficient use of disk drives, and better security.

The usual recommendation is swap size should be equal to physical memory in size; sometimes twice that is recommended. However, the correct choice depends on the related issues of system use scenarios as well as hardware capabilities.

Adding more and more swap will not necessarily help because at a certain point it becomes useless. One will need to either add more memory or re-evaluate the system setup.

## 17.8 Backing Up and Restoring Partition Tables

# Backing Up and Restoring Partition Tables

- **MBR**

- Use the **dd** program:
- The following command will backup the **MBR** (along with the partition table)
 

```
$ dd if=/dev/sda of=mbrbackup bs=512 count=1
```
- The **MBR** can be restored using the following
 

```
$ sudo dd if=mbrbackup of=/dev/sda bs=512 count=1
```

- **GPT**

- Use **sgdisk** which can also be used on **MBR** systems

```
$ sudo sgdisk --backup-file=gptbackup /dev/sda
$ sudo sgdisk --load-backup-file=gptbackup /dev/sda
```

Always assume changing the disk partition table might eliminate all data in all filesystems (It should not, but be cautious!).

Thus, it is always prudent to make a backup of all data (that is not already backed up) before doing any of this type of work. For **MBR** systems, **dd** can be used for converting and copying files. Be careful in using **dd**: A simple typing error or misused option could destroy your entire disk. The above **dd** commands only copy the primary partition table; they do not deal with any partition tables stored in the other partitions (for extended partitions, etc.)

For **GPT** systems it is best to use the **sgdisk** tool as in:

```
x8:/tmp>sudo sgdisk -p /dev/sda
```

```
1 Disk /dev/sda: 1000215216 sectors, 476.9 GiB
2 Model: SAMSUNG MZNLN512
3 Sector size (logical/physical): 512/512 bytes
4 Disk identifier (GUID): DBDBE747-8392-4B62-97AA-6214490073DC
5 Partition table holds up to 128 entries
6 ...
7 Number Start (sector) End (sector) Size Code Name
8 1 2048 534527 260.0 MiB EF00 EFI System Partition
9 2 534528 567295 16.0 MiB OC01 Microsoft reserved ...
10 ...
```

Note if run on a pure **MBR** system:

```
c8:/tmp>sudo sgdisk -p /dev/sda
```

```
1 ****
2 Found invalid GPT and valid MBR; converting MBR to GPT format
3 in memory.
4 ****
5 Disk /dev/sda: 500118192 sectors, 238.5 GiB
6 Model: Crucial_CT256MX1
7 Sector size (logical/physical): 512/4096 bytes
8
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 17.9 Partition table editors

# Partition Table Editors

- **fdisk**: standard, menu oriented
- **sfdisk**: script or command line interface
- **parted**: GNU partition manipulation program
- **gparted**: GUI based version of **parted**
- **gdisk**: For **GPT** systems; can also operate on **MBR** systems
- **sgdisk**: script or command line interface

**fdisk** is a menu driven partition table editor. It is the most standard and one of the most flexible of the partition table editors. As with any partition table editor, make sure that you either write down the current partition table settings or make a copy of the current settings before making changes.

The **sfdisk** command is a non-interactive **Linux**-based partition editor program, making it useful for scripting. Use the **sfdisk** tool with care.

For **GPT** systems, **gdisk** and **sgdisk** play a similar role. Both also work on **MBR** systems.

**parted** is the GNU partition manipulation program. It can create, remove, resize, and move partitions (including certain filesystems). The GUI interface to the **parted** command is **gparted**.

Many **Linux** distributions have a **live/installation** version which can be run off either a **CDROM** or **USB** stick, which include a copy of **gparted**, so they can easily be used as a graphical partitioning tool on disks which are not actually being used while the partitioning program is run.

## 17.10 fdisk

# Using fdisk

- Part of the base install, easy to use
- Menu oriented, just follow prompts:
  - m: Display the menu.
  - p: List the partition table.
  - n: Create a new partition.
  - d: Delete a partition.
  - t: Change a partition type.
  - w: Write the new partition table information and exit.
  - q: Quit without making changes.
- Edit the partition table:  
`$ sudo fdisk /dev/sda`
- Display the partition table and take no action:  
`$ sudo fdisk -l /dev/sda`

You must be root to run **fdisk**. It can be somewhat complex to handle and caution is advised. The **fdisk** interface is text-menu driven. After starting on a particular disk as in:

```
$ sudo fdisk /dev/sdb
```

you can issue the one-letter commands itemized above.

Fortunately, no actual changes are made until you write the partition table to the disk by entering **w**. It is therefore important to verify your partition table is correct (with **p**), before writing to disk with **w**. If something is wrong, you can jump out safely with **q**. The system will not use the new partition table until you reboot. However, you can use the following command:

```
$ sudo partprobe -s
```

to try and read in the revised partition table. However, this does not always work reliably and it is best to reboot before doing things like formatting new partitions etc as mixing up partitions can be catastrophic.

At any time you can do:

```
$ cat /proc/partitions
```

to examine what partitions the operating system is currently aware of.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 17.11 Labs



### Video Demonstration Resources

[using\\_fdisk\\_demo.mp4](#)

## Exercise 17.0: Using Loop Back Devices to Emulate Partitions

The exercises in this section will make simple use of **mkfs** for formatting filesystems, and **mount** for mounting them at places in the root filesystem tree. These commands will be explained in detail in the next session.

In the first three exercises in this section, we will use the **loop device** mechanism with or without the **parted** program.



### Very Important

For the purposes of later exercises in this course you will need unpartitioned disk space. It need not be large, certainly one or two GB will suffice.

If you are using your own native machine, you either have it or you do not. If you do not, you will have shrink a partition and the filesystem on it (first!) and then make it available, using **gparted** and/or the steps we have outlined or will outline.



### Please Note

If you have real physical unpartitioned disk space you do not **need** to do the following procedures, but it is still a very useful learning exercise.

## Exercise 17.1: Using a File as a Disk Partition Image

In this first exercise, we are going to create a file that will be used as a container for a full hard disk partition image, and for all intents and purposes can be used like a real hard partition. In the following exercise, we will show how to put more than one partition on it and have it behave as an entire disk.

1. Create a file full of zeros 1 GB in length:

```
$ dd if=/dev/zero of=imagefile bs=1M count=1024
```

You can make a much smaller file if you like or do not have that much available space in the partition you are creating the file on.

2. Put a filesystem on it:

```
$ mkfs.ext4 imagefile
```

```
1 mke2fs 1.42.9 (28-Dec-2013)
2 imagefile is not a block special device.
3 Proceed anyway? (y,n) y
4 Discarding device blocks: done
5
```

Of course you can format with a different filesystem, doing **mkfs.ext3**, **mkfs.vfat**, **mkfs.xfs** etc.

3. Mount it somewhere:

For the exclusive use of LFS301 corp class taught 06 to 09 December

```
$ mkdir mntpoint
$ sudo mount -o loop imagefile mntpoint
```

You can now use this to your heart's content, putting files etc. on it.

- When you are done unmount it with:

```
$ sudo umount mntpoint
```

An alternative method to using the `loop` option to mount would be:

```
$ sudo losetup /dev/loop2 imagefile
$ sudo mount /dev/loop2 mntpoint
...
$ sudo umount mntpoint
$ sudo losetup -d /dev/loop2
```

We will discuss `losetup` in a subsequent exercise, and you can use `/dev/loop[0-7]` but you have to be careful they are not already in use, as we will explain.

You should note that using a loop device file instead of a real partition can be useful, but it is pretty worthless for doing any kind of measurements or benchmarking. This is because you are placing one filesystem layer on top of another, which can only have a negative effect on performance, and mostly you just use the behavior of the underlying filesystem the image file is created on.

## Exercise 17.2: Partitioning a Disk Image File

The next level of complication is to divide the container file into multiple partitions, each of which can be used to hold a filesystem, or a swap area.

You can reuse the image file created in the previous exercise or create a new one.

- Run `fdisk` on your imagefile:

```
$ sudo fdisk -C 130 imagefile
1 Device does not contain a recognized partition table
2 Building a new DOS disk label with disk identifier 0x6280ced3.
3 Welcome to fdisk (util-linux 2.23.2).
4
5 Changes will remain in memory only, until you decide to write them.
6 Be careful before using the write command.
7
8 Command (m for help):
```

The `-C 130` sets the number of phony cylinders in the drive, and is only necessary in old versions of `fdisk`, which unfortunately you will find on **RHEL 6**. However, it will do no harm on other distributions.

- Type `m` to get a list of commands:

```
m
1 Command (m for help): m
2 Command action
3 a toggle a bootable flag
4 b edit bsd disklabel
5 c toggle the dos compatibility flag
6 d delete a partition
7 g create a new empty GPT partition table
8 G create an IRIX (SGI) partition table
9 l list known partition types
10 m print this menu
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

```

11 n add a new partition
12 o create a new empty DOS partition table
13 p print the partition table
14 q quit without saving changes
15 s create a new empty Sun disklabel
16 t change a partition's system id
17 u change display/entry units
18 v verify the partition table
19 w write table to disk and exit
20 x extra functionality (experts only)
21
22 Command (m for help):
```

3. Create a new primary partition and make it 256 MB (or whatever size you would like):

Command (m for help): n

```

1 Partition type:
2 p primary (0 primary, 0 extended, 4 free)
3 e extended
4 Select (default p): p
5 Partition number (1-4, default 1): 1
6 First sector (2048-2097151, default 2048):
7 Using default value 2048
8 Last sector, +sectors or +size{K,M,G} (2048-2097151, default 2097151): +256M
9 Partition 1 of type Linux and of size 256 MiB is set
```

4. Add a second primary partition also of 256 MB in size:

Command (m for help): n

```

1 Partition type:
2 p primary (1 primary, 0 extended, 3 free)
3 e extended
4 Select (default p): p
5 Partition number (2-4, default 2): 2
6 First sector (526336-2097151, default 526336):
7 Using default value 526336
8 Last sector, +sectors or +size{K,M,G} (526336-2097151, default 2097151): +256M
9 Partition 2 of type Linux and of size 256 MiB is set
10
11 Command (m for help): p
12
13 Disk imagefile: 1073 MB, 1073741824 bytes, 2097152 sectors
14 Units = sectors of 1 * 512 = 512 bytes
15 Sector size (logical/physical): 512 bytes / 512 bytes
16 I/O size (minimum/optimal): 512 bytes / 512 bytes
17 Disk label type: dos
18 Disk identifier: 0x6280ced3
19
20 Device Boot StartEnd Blocks Id System
21 imagefile1 2048 526335 262144 83 Linux
22 imagefile2 526336 1050623 262144 83 Linux
```

5. Write the partition table to disk and exit:

Command (m for help): w

```

1 The partition table has been altered!
2
3 Syncing disks.
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

While this has given us some good practice, we haven't yet seen a way to use the two partitions we just created. We'll start over in the next exercise with a method that lets us do so.

## Exercise 17.3: Using losetup and parted

We are going to experiment more with:

- Loop devices and **losetup**
- **parted** to partition at the command line non-interactively.

We expect that you should read the **man pages** for **losetup** and **parted** before doing the following procedures.

Once again, you can reuse the image file or, better still, zero it out and start freshly or with another file.

1. Associate the image file with a **loop** device:

```
$ sudo losetup -f
1 /dev/loop1
```

```
$ sudo losetup /dev/loop1 imagefile
```

where the first command finds the first **free** loop device. The reason to do this is you may already be using one or more loop devices. For example, on the system that this is being written on, before the above command is executed:

```
$ losetup -a
1 /dev/loop0: []: (/usr/src/KERNELS.sqfs)
```

a **squashfs** compressed, read-only filesystem is already mounted using **/dev/loop0**. (The output of this command will vary with distribution.) If we were to ignore this and use **losetup** on **/dev/loop0** we would almost definitely corrupt the file.

2. Create a disk partition label on the loop device (image file):

```
$ sudo parted -s /dev/loop1 mklabel msdos
```

3. Create three primary partitions on the loop device:

```
$ sudo parted -s /dev/loop1 unit MB mkpart primary ext4 0 256
$ sudo parted -s /dev/loop1 unit MB mkpart primary ext4 256 512
$ sudo parted -s /dev/loop1 unit MB mkpart primary ext4 512 1024
```

4. Check the partition table:

```
$ fdisk -l /dev/loop1
1 Disk /dev/loop1: 1073 MB, 1073741824 bytes, 2097152 sectors
2 Units = sectors of 1 * 512 = 512 bytes
3 Sector size (logical/physical): 512 bytes / 512 bytes
4 I/O size (minimum/optimal): 512 bytes / 512 bytes
5 Disk label type: dos
6 Disk identifier: 0x00050c11
7
8 Device Boot Start End Blocks Id System
9 /dev/loop1p1 1 500000 250000 83 Linux
10 /dev/loop1p2 500001 1000000 250000 83 Linux
11 /dev/loop1p3 1000001 2000000 500000 83 Linux
```

5. What happens next depends on what distribution you are on. For example, on **RHEL** and **Ubuntu** you will find new device nodes have been created:

For the exclusive use of LFS301 corp class taught 06 to 09 December

```
$ ls -l /dev/loop1*
1 brw-rw---- 1 root disk 7, 1 Oct 7 14:54 /dev/loop1
2 brw-rw---- 1 root disk 259, 0 Oct 7 14:54 /dev/loop1p1
3 brw-rw---- 1 root disk 259, 3 Oct 7 14:54 /dev/loop1p2
4 brw-rw---- 1 root disk 259, 4 Oct 7 14:54 /dev/loop1p3
```

and we will use them in the following.

6. Put filesystems on the partitions:

```
$ sudo mkfs.ext3 /dev/loop1p1
$ sudo mkfs.ext4 /dev/loop1p2
$ sudo mkfs.vfat /dev/loop1p3
```

7. Mount all three filesystems and show they are available:

```
$ mkdir mnt1 mnt2 mnt3

$ sudo mount /dev/loop1p1 mnt1
$ sudo mount /dev/loop1p2 mnt2
$ sudo mount /dev/loop1p3 mnt3
```

```
$ df -Th
1 Filesystem Type Size Used Avail Use% Mounted on
2 /dev/sda1 ext4 29G 8.5G 19G 32% /
3
4 /dev/loop1p1 ext3 233M 2.1M 219M 1% mnt1
5 /dev/loop1p2 ext4 233M 2.1M 215M 1% mnt2
6 /dev/loop1p3 vfat 489M 0 489M 0% mnt3
```

8. After using the filesystems to your heart's content you can unwind it all:

```
$ sudo umount mnt1 mnt2 mnt3
$ rmdir mnt1 mnt2 mnt3
$ sudo losetup -d /dev/loop1
```

## Exercise 17.4: Partitioning a Real Hard Disk

If you have real hard disk un-partitioned space available, experiment with **fdisk** to create new partitions, either primary or logical within an extended partition. Write the new partition table to disk and then format and mount the new partitions.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 18

# Filesystem Features: Attributes, Creating, Checking, Mounting



|      |                                               |     |
|------|-----------------------------------------------|-----|
| 18.1 | Extended Attributes . . . . .                 | 224 |
| 18.2 | Creating and formatting filesystems . . . . . | 225 |
| 18.3 | Checking and Repairing Filesystems . . . . .  | 226 |
| 18.4 | Mounting filesystems . . . . .                | 227 |
| 18.5 | NFS . . . . .                                 | 231 |
| 18.6 | Mounting at Boot and /etc/fstab . . . . .     | 232 |
| 18.7 | automount . . . . .                           | 234 |
| 18.8 | Labs . . . . .                                | 236 |

## 18.1 Extended Attributes

# lsattr and chattr

- Extended attributes may be set for files
  - i: immutable flag
  - a: append-only flag
  - d: no dump flag
  - A: No atime update flag
- Flags are viewed with **lsattr** command
- Flags are set with **chattr** command
- Flags are stored in the file's inode flags
- May only be set by root

**Extended Attributes** associate metadata not interpreted directly by the filesystem with files. Four **namespaces** exist: user, trusted, security and system. The system namespace is used for **access control lists (ACLs)** and the security namespace is used by **SELinux**.

Flag values are stored in the file inode and may be modified and set only by the root user. They are viewed with **lsattr** and set with **chattr**. Flags may be set for files:

- **i (immutable)**: Cannot be modified (not even by root), or deleted or renamed. No hard link can be created to it, and no data can be written to the file. Only the superuser can set or clear this attribute.
- **a (append-only)**: Can only be opened in append mode for writing. Only the superuser can set or clear this attribute.
- **d (no-dump)**: Is ignored when the **dump** program is run. This is useful for swap and cache files that you do not want to waste time backing up.
- **A (No atime update)**: The system will not modify the file's **atime** (access time) record when the file is accessed but not otherwise modified. This can increase the performance on some systems because it reduces the amount of disk I/O.

Note that there are other flags that can be set; typing `man chattr` will show the whole list.

The format for **chattr** is:

```
$ chattr [+|-|=mode] filename
```

**lsattr** is used to display attributes for a file:

```
$ lsattr filename
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 18.2 Creating and formatting filesystems

### mkfs

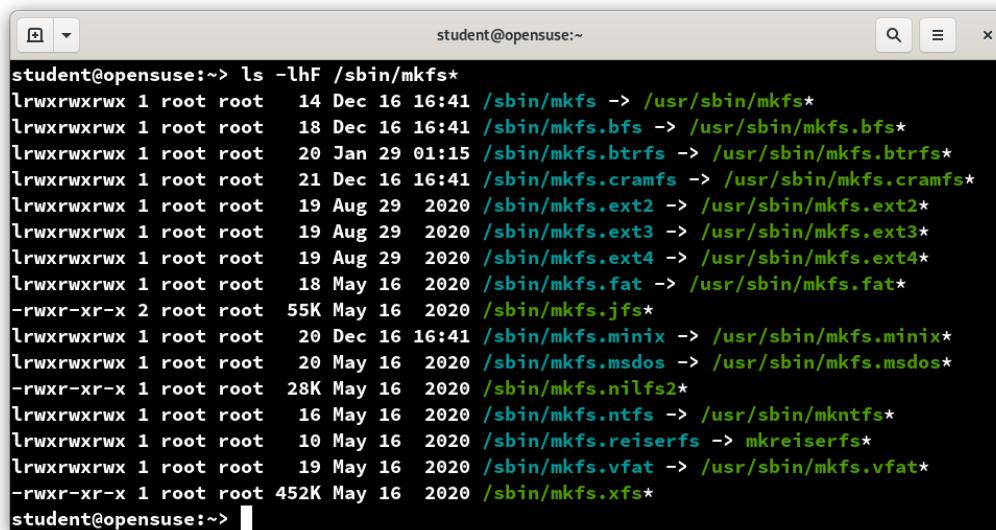
- General format for **mkfs**:

```
mkfs [-t fstype] [options] [device-file]
```

- [**device-file**] is usually a device name like `/dev/sda3` or `/dev/vg/lvm1`.
- Each filesystem type has its own particular formatting options
- Each filesystem has its own **mkfs** program
- Equivalent commands:

```
$ sudo mkfs -t ext4 /dev/sda10
$ sudo mkfs.ext4 /dev/sda10
```

Every filesystem type has a utility for **formatting** (making) a filesystem on a partition. The generic name for these utilities is **mkfs**. However **mkfs** is just a front-end for filesystem-specific programs, each of which may have particular options:



```
student@opensuse:~> ls -lhF /sbin/mkfs*
lrwxrwxrwx 1 root root 14 Dec 16 16:41 /sbin/mkfs -> /usr/sbin/mkfs*
lrwxrwxrwx 1 root root 18 Dec 16 16:41 /sbin/mkfs.bfs -> /usr/sbin/mkfs.bfs*
lrwxrwxrwx 1 root root 20 Jan 29 01:15 /sbin/mkfs.btrfs -> /usr/sbin/mkfs.btrfs*
lrwxrwxrwx 1 root root 21 Dec 16 16:41 /sbin/mkfs.cramfs -> /usr/sbin/mkfs.cramfs*
lrwxrwxrwx 1 root root 19 Aug 29 2020 /sbin/mkfs.ext2 -> /usr/sbin/mkfs.ext2*
lrwxrwxrwx 1 root root 19 Aug 29 2020 /sbin/mkfs.ext3 -> /usr/sbin/mkfs.ext3*
lrwxrwxrwx 1 root root 19 Aug 29 2020 /sbin/mkfs.ext4 -> /usr/sbin/mkfs.ext4*
lrwxrwxrwx 1 root root 18 May 16 2020 /sbin/mkfs.fat -> /usr/sbin/mkfs.fat*
-rw-r-xr-x 2 root root 55K May 16 2020 /sbin/mkfs.jfs*
lrwxrwxrwx 1 root root 20 Dec 16 16:41 /sbin/mkfs.minix -> /usr/sbin/mkfs.minix*
lrwxrwxrwx 1 root root 20 May 16 2020 /sbin/mkfs.msdos -> /usr/sbin/mkfs.msdos*
-rw-r-xr-x 1 root root 28K May 16 2020 /sbin/mkfs.nilfs2*
lrwxrwxrwx 1 root root 16 May 16 2020 /sbin/mkfs.ntfs -> /usr/sbin/mkntfs*
lrwxrwxrwx 1 root root 10 May 16 2020 /sbin/mkfs.reiserfs -> mkreiserfs*
lrwxrwxrwx 1 root root 19 May 16 2020 /sbin/mkfs.vfat -> /usr/sbin/mkfs.vfat*
-rw-r-xr-x 1 root root 452K May 16 2020 /sbin/mkfs.xfs*
```

Figure 18.1: **mkfs**

One should look at the **man** page for each of the `mkfs.*` programs to see details.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 18.3 Checking and Repairing Filesystems

### fsck

- Run automatically after a set period of time or number of mounts
- Run at boot when not unmounted cleanly previously
- Should not be run on mounted filesystems
- General format for **fsck**:

```
fsck [-t fstype] [options] [device-file]
```

where [device-file] is usually a device name like `/dev/sda3` or `/dev/vg/lvm1`.

- Filesystem type usually not needed as partition superblock is examined
- Equivalent commands:

```
$ sudo fsck -t ext4 /dev/sda10
$ sudo fsck.ext4 /dev/sda10
```

Every filesystem type has a utility designed to check for errors (and hopefully fix any that are found). The generic name for these utilities is **fsck**. However this is just a front-end for filesystem-specific programs:

```
student@opensuse:~> ls -lhF /sbin/fsck*
lrwxrwxrwx 1 root root 14 Dec 16 16:41 /sbin/fsck -> /usr/sbin/fsck*
lrwxrwxrwx 1 root root 20 Jan 29 01:15 /sbin/fsck.btrfs -> /usr/sbin/fsck.btrfs*
lrwxrwxrwx 1 root root 21 Dec 16 16:41 /sbin/fsck.cramfs -> /usr/sbin/fsck.cramfs*
lrwxrwxrwx 1 root root 19 Aug 29 2020 /sbin/fsck.ext2 -> /usr/sbin/fsck.ext2*
lrwxrwxrwx 1 root root 19 Aug 29 2020 /sbin/fsck.ext3 -> /usr/sbin/fsck.ext3*
lrwxrwxrwx 1 root root 19 Aug 29 2020 /sbin/fsck.ext4 -> /usr/sbin/fsck.ext4*
lrwxrwxrwx 1 root root 18 May 16 2020 /sbin/fsck.fat -> /usr/sbin/fsck.fat*
-rw-r--r-- 2 root root 407K May 16 2020 /sbin/fsck.jfs*
lrwxrwxrwx 1 root root 20 Dec 16 16:41 /sbin/fsck.minix -> /usr/sbin/fsck.minix*
lrwxrwxrwx 1 root root 20 May 16 2020 /sbin/fsck.msdos -> /usr/sbin/fsck.msdos*
lrwxrwxrwx 1 root root 10 May 16 2020 /sbin/fsck.reiserfs -> reiserfsck*
lrwxrwxrwx 1 root root 19 May 16 2020 /sbin/fsck.vfat -> /usr/sbin/fsck.vfat*
-rw-r--r-- 1 root root 433 May 16 2020 /sbin/fsck.xfs*
```

Figure 18.2: **fsck**

You can force a check of all mounted filesystems at boot by doing:

```
$ sudo touch /forcefsck
$ sudo reboot
```

`/forcefsck` will disappear after the successful check.

One can control whether any errors found should be fixed one by one manually with the `-r` option, or automatically as best possible by using the `-a` option. In addition, each filesystem type may have its own particular `fsck` options.

For the exclusive use of LFS301 corp class taught 08 to 09 December

## 18.4 Mounting filesystems

# Mounting Filesystems

- Each filesystem is mounted under a specific directory:

```
$ mount -t ext4 /dev/sdb4 /home
```

- Mounts an **ext4** filesystem
- Usually not necessary to specify the type with **-t** option
- The filesystem is located on a specific partition of a hard drive (**/dev/sdb4**)
- The filesystem is mounted at the position **/home** in the current directory tree
- Any files residing in the original **/home** directory are hidden until the partition is unmounted

All accessible files in **Linux** are organized into one large hierarchical tree structure with the head of the tree being the root directory (**/**). However, it is common to have more than one partition (each of which can have its own filesystem type) joined together in the same filesystem tree. These partitions can also be on different physical devices, even on a network.

The **mount** program allows attaching at any point in the tree structure; **umount** allows detaching them.

The **mount point** is the directory where the filesystem is attached. It must exist before **mount** can use it; **mkdir** can be used to create an empty directory. If a pre-existing directory is used and it contains files prior to being used as a mount point, they will be hidden after mounting. These files are not deleted and will again be visible when the filesystem is unmounted.

Only the superuser can mount and unmount filesystems.

# mount

- General form for **mount**

```
mount [options] <source> <directory>
```

- Can mount using: **Device Node**, **Label** or **UUID**
- Each filesystem has its own set of options. A common example would be:

```
$ sudo mount -o remount,ro /myfs
```

which remounts a filesystem with a read-only attribute.

```
File Edit View Search Terminal Help
c:/tmp> mount --help
Usage:
mount [-lhV]
mount -a [options]
mount [options] [->source] <source> | [--target] <directory>
mount [options] <source> <directory>
mount <operation> <mountpoint> [<target>]

Options:
-a, --all mount all filesystems mentioned in fstab
-c, --no-canonicalize don't canonicalize paths
-f, --fake dry run; skip the mount(2) syscall
-F, --fork fork off for each device (use with -a)
-T, --fstab <path> alternative file to /etc/fstab
-h, --help display this help text and exit
-i, --internal-only don't call the mount.<type> helpers
-l, --show-labels lists all mounts with LABELS
-n, --noatime don't write to fattr
-o, --options <list> comma-separated list of mount options
-O, --test-opts <list> limit the set of filesystems (use with -a)
-r, --read-only mount the filesystem read-only (same as -o ro)
-t, --type <list> limit the set of filesystem types
--source <src> explicitly specifies source (path, label, uuid)
--target <target> explicitly specifies mountpoint
-v, --verbose say what is being done
-V, --version display version information and exit
-w, --rw, --read-write mount the filesystem read-write (default)

-h, --help display this help and exit
-V, --version output version information and exit
...
c:/tmp>
```

Figure 18.3: **mount**

The following ways of mounting are all equivalent:

```
$ sudo mount /dev/sda2 /home
$ sudo mount LABEL=home /home
$ sudo mount -L home /home
$ sudo mount UUID=26d58ee2-9d20-4dc7-b6ab-aa87c3cfb69a /home
$ sudo mount -U 26d58ee2-9d20-4dc7-b6ab-aa87c3cfb69a /home
```

Labels are assigned by filesystem type specific utilities, such as **e2label**, and **UUIDs** are assigned when partitions are created as containers for the filesystem, and formatted with **mkfs**.

While any of these three methods for specifying the device can be used, modern systems deprecate using the device node form because the names can change according to how the system is booted, which hard drives are found first, etc.

Labels are an improvement, but on rare occasions one could have two partitions that wind up with the same label. **UUIDs**, however, should always be unique and are created when the partitions are created.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Currently Mounted Filesystems

```
*mountout
/tmp
Save _ x
c7:/tmp>mount

sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
devtmpfs on /dev type devtmpfs (rw,nosuid,size=8124168k,nr_inodes=2031042,mode=755)
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)alai
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,nodev,mode=755)
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,mode=755)
cgroup on /sys/fs/cgroup/systemd type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,release_agent=/usr/lib/sys
pstree on /sys/fs/pstore type pstore (rw,nosuid,nodev,noexec,relatime)
.....
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime,pagesize=2M)
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
fusectl on /sys/fs/fuse/connections type fusectl (rw,relatime)
/dev/sdcl on /VM8 type ext4 (rw,relatime,stripe=32744)
/dev/sdb1 on /RHELT7 type ext4 (rw,relatime)
/dev/sdb2 on /DEAD type ext4 (rw,relatime,stripe=32743)
/dev/sdb2 on /usr/src type ext4 (rw,relatime,stripe=32743)
/dev/sdb2 on /usr/local type ext4 (rw,relatime,stripe=32743)
/usr/src/KERNELS.sqfs on /usr/src/KERNELS type squashfs (ro,relatime)
/dev/mapper/VG2-pictures on /PICTURES type ext4 (rw,relatime)
/dev/mapper/VG2-iso Images on /ISO IMAGES type ext4 (rw,relatime,stripe=32717)
/dev/mapper/VG2-audio on /AUDIO type ext4 (rw,relatime)
/dev/mapper/VG2-virtual on /VIRTUAL type ext4 (rw,relatime,stripe=32745)
/dev/mapper/VG2-dead2 on /DEAD2 type ext4 (rw,relatime,stripe=32698)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw,relatime)
vmware-vmblock on /run/vmblock-fuse type fuse.vmware-vmblock (rw,nosuid,nodev,relatime,user_id=0,group_id=0,de
tmpfs on /run/user/42 type tmpfs (rw,nosuid,nodev,relatime,size=1627508k,mode=700,uid=42,gid=42)
tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,relatime,size=1627508k,mode=700,uid=1000,gid=1000)
gvfsd-fuse on /run/user/1000/gvfs type fuse.gvfsd-fuse (rw,nosuid,nodev,relatime,user_id=1000,group_id=1000)
```

**Figure 18.4: Currently Mounted Filesystems**

# umount

- Used to unmount filesystems

- General form of **umount**:

```
umount [device-file | mount-point]
```

- Examples:

- Unmount the `/home` filesystem:

```
$ umount /home
```

- Unmount `/dev/sda3` device:

```
$ umount /dev/sda3
```

Note the command to unmount a filesystem is **umount** (not **unmount!**).

Like **mount**, **umount** has many options, many of which are specific to filesystem type. Once again, the **man** pages are the best source for specific option information.

The most common error encountered when unmounting a filesystem is trying to do this on a filesystem currently in use; i.e., there are current applications using files or other entries in the filesystem.

This can be as simple as having a terminal window open in a directory on the mounted filesystem. Just using **cd** in that window, or killing it, will get rid of the `device is busy` error and allow unmounting.

However, if there are other processes inducing this error, you must kill them before unmounting the filesystem. You can use **fuser** to find out which users are using the filesystem and kill them (be careful with this, you may also want to warn users first). You can also use **lsof** (“list open files”) to try and see which files are being used and blocking unmounting.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 18.5 NFS

# Network Shares (NFS)

- Most common network share is **NFS**
- Others include **AFS** and **SMB (CIFS)**
- **mount** as in:  

```
$ sudo mount -t nfs myserver.com:/shdir /mnt/shdir
```
- Put this line in **/etc/fstab** to mount on boot or with **mount -a**:  

```
myserver.com:/shdir /mnt/shdir nfs rsize=8192,wszie=8192,timeo=14,intr 0 0
```
- System may try to mount **NFS** filesystem before the network is up. Can use **netdev** and **noauto** options (**man nfs**, examine **mount** options)
- Can be solved also using **autofs** or **automount**
- **mount** has a million options, some specific to **nfs**. See the **man** pages for both **nfs** and **mount** for more detail.

It is common to mount remote filesystems through network shares, so they appear as if they were on the local machine.

Probably the most common method used historically has been **NFS (Network File System)**.

**NFS** was originally developed by **Sun Microsystems** in 1989, and has been continuously updated. Modern systems use **NFSv4**, which has been continuously updated since 2000.

Other network filesystems include **AFS (Andrew File System)** and **SMB (Server Message Block)**, also termed **CIFS**.

Because a network file system may either be unavailable at any time, either because it is not present on the network share, or the network is unavailable, systems have to be prepared for this possibility.

Thus, in such circumstances, a system should be instructed not to get hung, or blocked, while waiting longer than a specified period. These can be specified in the **mount** command or in **/etc/fstab**.

## 18.6 Mounting at Boot and /etc/fstab

### Mounting at Boot

- During system initialization the command:

```
$ mount -a
```

is executed in order to mount all filesystems listed in `/etc/fstab`

- Can specify local and remote network-mounted filesystems to be mounted at bootup

`/etc/fstab` is used to define mountable file systems and devices on startup. Look there to see what file systems can be mounted, where they may be found locally, who may mount them and with what permissions. If a file or directory is specified in `/etc/fstab`, it may be mounted at startup.

This may include both local and remote network-mounted filesystems, such as NFS and samba filesystems.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## /etc/fstab

- Contains information about filesystems, their standard mount points and what options should be used when mounting them.
- Each record in the file contains white space separated fields of information about a filesystem to be mounted
  - Device-file, label, or UUID
  - Mount point
  - Filesystem type
  - A comma-separated options list
  - Dump frequency (or a zero)
  - **fsck** pass number (or a zero)
- The **mount** and **umount** commands can use information in */etc/fstab*

Each record in the file contains information about a filesystem to be mounted at boot time. The first item in the record may either be a device-file name (such as */dev/sda1*), a label, or a UUID . This is followed by the mount point for the filesystem (where in the tree structure is it to be inserted). Then comes the filesystem type followed by a comma-separated list of options, followed by the dump frequency (used by the **dump -w** command) or a zero (ignored by **dump**), followed by the **fsck** pass number or a zero (meaning do not **fsck** this partition).

```
x8:/tmp> cat /etc/fstab
#
/etc/fstab
Created by anaconda on Tue Aug 27 15:29:19 2019
#
Accessible filesystems, by reference, are maintained under '/dev/disk/'.
See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info.
#
After editing this file, run 'systemctl daemon-reload' to update systemd
units generated from this file.
#
UUID=53ea9807-fd58-1234-9460-d03ec36f73a3 / ext4 defaults 1 1
UUID=29c4-7777 /boot/efi vfat umask=0077,shortname=winnt 0 2
LABEL=ALL /ALL ext4 defaults 1 2
LABEL=UBUNTU /UBUNTU ext4 defaults 1 2

/ALL/usr/local /usr/local none bind 0 0
/ALL/usr/src /usr/src none bind 0 0
/ALL/VMS /VMS none bind 0 0
/ALL/RESOURCES /RESOURCES none bind 0 0

LABEL=Sam128 /SAM ext4 noauto 0 0
/usr/src/KERNELS.sqfs /usr/src/KERNELS squashfs loop 0 0

/dev/sda1 /Cbootefi vfat umask=0077,shortname=winnt 0 2
x8:/tmp>
```

Figure 18.5: /etc/fstab Example  
For the exclusive use of LFS301 corp class taught 06 to 09 December

## 18.7 automount

# Automatic Filesystem Mounting

- Make filesystem available on use (**mount**), remove on idle (**umount**)
- **autofs**:
  - Available for many years,
  - Requires installation of **autofs** package
  - Flexible, configuration files in `/etc`
- **automount**
  - Part of **systemd**
  - Modify `/etc/fstab`, and reboot or restart **systemd** as in:

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart local-fs.target
```

**Linux** systems have long had the ability to mount a filesystem only when it is needed. Historically, this was done using **autofs**. This utility requires installation of the **autofs** package using the appropriate package manager and configuration of files in `/etc`.

While **autofs** is very flexible and well understood, **systemd**-based systems (including all enterprise **Linux** distributions) come with **automount** facilities built into the **systemd** framework. Configuring this is as simple as adding a line in `/etc/fstab` specifying the proper device, mount point and mounting options, such as:

```
LABEL=Sam128 /SAM ext4 noauto,x-systemd.automount,x-systemd.device-timeout=10,x-systemd.idle-timeout=30 0 0
```

and then either rebooting or issuing the command:

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart local-fs.target
```

On the next page we will give an example and explain the options.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## automount Example

```

File Edit View Search Terminal Help
x7:/tmp>grep automount /etc/fstab
LABEL=Sam128 /SAM ext4 noauto,x-systemd.automount,x-systemd.device-tim
eout=10,x-systemd.idle-timeout=30 0 0
x7:/tmp>df -h | grep SAM
x7:/tmp>ls /SAM
ISO_IMAGES lost+found VIRTUAL_MACHINE_IMAGES
x7:/tmp>df -h | grep SAM
/dev/sdb1 ext4 118G 71G 41G 64% /SAM
x7:/tmp>sleep 40
x7:/tmp>df -h | grep SAM
x7:/tmp>

```

Figure 18.6: automount Example

The above example mounts a **USB** pen drive that is always plugged into the system, only when it is used.

Options in `/etc/fstab`:

- `noauto`  
Do not mount at boot. Here auto does not refer to **automount**.
- `x-systemd.automount`  
Use the **systemd automount** facility
- `x-systemd.automount.device-timeout=10`  
If this device is not available, say it is a network device accessible through **NFS**, timeout after 10 seconds instead of getting hung.
- `x-systemd.automount.idle-timeout=30`  
If the device is not used for 30 seconds, unmount it.

Note that the device **may** be mounted during boot, but then it should be unmounted after the timeout specified.

The screenshot shows how the device is only available once it is used.

## 18.8 Labs



### Video Demonstration Resources

[using\\_filesystems\\_demo.mp4](#)

#### Exercise 18.1: Working with File Attributes

1. With your normal user account use **touch** to create an empty file named **/tmp/appendit**.
2. Use **cat** to append the contents of **/etc/hosts** to **/tmp/appendit**.
3. Compare the contents of **/tmp/appendit** with **/etc/hosts**; there should not be any differences.
4. Try to add the append-only attribute to **/tmp/appendit** by using **chattr**. You should see an error here. Why?
5. As root, retry adding the append-only attribute; this time it should work. Look at the file's extended attributes by using **lsattr**.
6. As a normal user, try and use **cat** to copy over the contents of **/etc/passwd** to **/tmp/appendit**. You should get an error. Why?
7. Try the same thing again as root. You should also get an error. Why?
8. As the normal user, again use the append redirection operator (**>>**) and try appending the **/etc/passwd** file to **/tmp/appendit**. This should work. Examine the resulting file to confirm.
9. As root, set the immutable attribute on **/tmp/appendit**, and look at the extended attributes again.
10. Try appending output to **/tmp/appendit**, try renaming the file, creating a hard link to the file, and deleting the file as both the normal user and as root.
11. We can remove this file by removing the extended attributes. Do so.

#### Solution 18.1

1. 

```
$ cd /tmp
$ touch appendit
$ ls -l appendit
```

1 -rw-rw-r-- 1 coop coop 0 Oct 23 19:04 appendit
2. 

```
$ cat /etc/hosts > appendit
```
3. 

```
$ diff /etc/hosts appendit
```
4. 

```
$ chattr +a appendit
```

1 chattr: Operation not permitted while setting flags on appendit
5. 

```
$ sudo chattr +a appendit
$ lsattr appendit
```

1 -----a-----e-- appendit
6. 

```
$ cat /etc/passwd > appendit
```

1 bash: appendit: Operation not permitted

For the exclusive use of LFS301 corp class taught 06 to 09 December

```
7. $ sudo su
$ cat /etc/passwd > appendit
```

```
1 bash: appendit: Operation not permitted
```

```
$ exit
```

```
8. $ cat /etc/passwd >> /tmp/appendit
$ cat appendit
```

```
9. $ sudo chattr +i appendit
$ lsattr appendit
```

```
1 -----ia-----e- appendit
```

```
10. $ echo hello >> appendit
```

```
1 -bash: appendit: Permission denied
```

```
$ mv appendit appendit.rename
```

```
1 mv: cannot move `appendit' to `appendit.rename': Operation not permitted
```

```
$ ln appendit appendit.hardlink
```

```
1 ln: creating hard link `appendit.hardlink' => `appendit': Operation not permitted
```

```
$ rm -f appendit
```

```
1 rm: cannot remove `appendit': Operation not permitted
```

```
$ sudo su
```

```
$ echo hello >> appendit
```

```
1 -bash: appendit: Permission denied
```

```
$ mv appendit appendit.rename
```

```
1 mv: cannot move `appendit' to `appendit.rename': Operation not permitted
```

```
$ ln appendit appendit.hardlink
```

```
1 ln: creating hard link `appendit.hardlink' => `appendit': Operation not permitted
```

```
$ rm -f appendit
```

```
1 rm: cannot remove `appendit': Operation not permitted
```

```
$ exit
```

```
11. $ sudo su
$ lsattr appendit
```

```
1 -----ia-----e- appendit
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

```
$ chattr -ia appendit
$ rm appendit
1 rm: remove regular file `appendit'? y
```

```
$ ls appendit
1 ls: cannot access appendit: No such file or directory
```

## Exercise 18.2: Mounting Options

### Fresh Partition or Loopback File

In this exercise you will need to either create a fresh partition, or use a loopback file. The solution will differ slightly and we will provide details of both methods.

1. Use **fdisk** to create a new 250 MB partition on your system, probably on `/dev/sda`. Or create a file full of zeros to use as a loopback file to simulate a new partition.
2. Use **mkfs** to format a new filesystem on the partition or loopback file just created. Do this three times, changing the block size each time. Note the locations of the superblocks, the number of block groups and any other pertinent information, for each case.
3. Create a new subdirectory (say `/mnt/tempdir`) and mount the new filesystem at this location. Verify it has been mounted.
4. Unmount the new filesystem, and then remount it as read-only.
5. Try to create a file in the mounted directory. You should get an error here, why?
6. Unmount the filesystem again.
7. Add a line to your `/etc/fstab` file so that the filesystem will be mounted at boot time.
8. Mount the filesystem.
9. Modify the configuration for the new filesystem so that binary files may not be executed from the filesystem (change defaults to `noexec` in the `/mnt/tempdir` entry). Then remount the filesystem and copy an executable file (such as `/bin/ls`) to `/mnt/tempdir` and try to run it. You should get an error: why?

When you are done you will probably want to clean up by removing the entry from `/etc/fstab`.

### Solution 18.2



#### Physical Partition Solution

1. We won't show the detailed steps in **fdisk**, as it is all ground covered earlier. We will assume the partition created is `/dev/sda11`, just to have something to show.

```
$ sudo fdisk /dev/sda
1
2 w
3 $ partprobe -s
```

Sometimes the **partprobe** won't work, and to be sure the system knows about the new partition you have to reboot.

For the exclusive use of LFS301 corp class taught 06 to 09 December

```
2. $ sudo mkfs -t ext4 -v /dev/sda11
$ sudo mkfs -t ext4 -b 2048 -v /dev/sda11
$ sudo mkfs -t ext4 -b 4096 -v /dev/sda11
```

Note the `-v` flag (verbose) will give the requested information; you will see that for a small partition like this the default is 1024 byte blocks.

```
3. $ sudo mkdir /mnt/tempdir
$ sudo mount /dev/sda11 /mnt/tempdir
$ mount | grep tempdir

4. $ sudo umount /mnt/tempdir
$ sudo mount -o ro /dev/sda11 /mnt/tempdir
```

If you get an error while unmounting, make sure you are not currently in the directory.

```
5. $ sudo touch /mnt/tempdir/afile
```

```
6. $ sudo umount /mnt/tempdir
```

7. Put this line in `/etc/fstab`:

```
/dev/sda11 /mnt/tempdir ext4 defaults 1 2
```

```
8. $ sudo mount /mnt/tempdir
$ sudo mount | grep tempdir
```

9. Change the line in `/etc/fstab` to:

```
/dev/sda11 /mnt/tempdir ext4 noexec 1 2
```

Then do:

```
$ sudo mount -o remount /mnt/tempdir
$ sudo cp /bin/ls /mnt/tempdir
$ /mnt/tempdir/ls
```

You should get an error here, why?



### Loopback File Solution

```
1. $ sudo dd if=/dev/zero of=/imagefile bs=1M count=250

2. $ sudo mkfs -t ext4 -v
$ sudo mkfs -t ext4 -b 2048 -v /imagefile
$ sudo mkfs -t ext4 -b 4096 -v /imagefile
```

You will get warned that this is a file and not a partition, just proceed.

Note the `-v` flag (verbose) will give the requested information; you will see that for a small partition like this the default is 1024 byte blocks.

```
3. $ sudo mkdir /mnt/tempdir
$ sudo mount -o loop /imagefile /mnt/tempdir
$ mount | grep tempdir

4. $ sudo umount /mnt/tempdir
$ sudo mount -o ro,loop /imagefile /mnt/tempdir
```

If you get an error while unmounting, make sure you are not currently in the directory.

```
5. $ sudo touch /mnt/tempdir/afile
```

```
6. $ sudo umount /mnt/tempdir
```

7. Put this line in `/etc/fstab`:

For the exclusive use of LFS301 corp class taught 06 to 09 December

**in /etc/fstab**

```
/imagefile /mnt/tempdir ext4 loop 1 2
```

8. \$ sudo mount /mnt/tempdir  
\$ sudo mount | grep tempdir

9. Change the line in **/etc/fstab** to:

**in /etc/fstab**

```
/imagefile /mnt/tempdir ext4 loop,noexec 1 2
```

Then do:

```
$ sudo mount -o remount /mnt/tempdir
$ sudo cp /bin/ls /mnt/tempdir
$ /mnt/tempdir/ls
```

You should get an error here, why?

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 19

# Filesystem Features: Swap, Quotas, Usage



|      |                                |     |
|------|--------------------------------|-----|
| 19.1 | Filesystem Usage . . . . .     | 242 |
| 19.2 | Disk Usage . . . . .           | 243 |
| 19.3 | Swap . . . . .                 | 244 |
| 19.4 | Filesystem Quotas ** . . . . . | 245 |
| 19.5 | Labs . . . . .                 | 251 |



### Please Note

\*\* These sections may be considered in part or in whole as optional. They contain either background reference material, specialized topics, or advanced subjects. The instructor may choose to cover or not cover them depending on classroom experience and time constraints.

## 19.1 Filesystem Usage

# df: Filesystem Usage

- **df** (disk free) is used to look at filesystem usage

- Display filesystem usage (in K bytes-default):

```
$ df
```

- Display filesystem usage in **human-readable** format:

```
$ df -h
```

- Display filesystem type format:

```
$ df -T
```

- Show inode information:

```
$ df -i
```

```
x8:/tmp>df -hT
Filesystem Type Size Used Avail Use% Mounted on
devtmpfs devtmpfs 7.8G 0 7.8G 0% /dev
tmpfs tmpfs 7.8G 0 7.8G 0% /dev/shm
tmpfs tmpfs 7.8G 9.6M 7.8G 1% /run
tmpfs tmpfs 7.8G 0 7.8G 0% /sys/fs/cgroup
/dev/sda8 ext4 26G 10G 15G 42% /
/dev/sda7 vfat 200M 14M 187M 7% /boot/efi
/dev/sda5 ext4 15G 9.6G 4.1G 71% /UBUNTU
/dev/sda6 ext4 389G 252G 118G 69% /ALL
/dev/sda1 vfat 256M 43M 214M 17% /Cbootefi
tmpfs tmpfs 1.6G 1.2M 1.6G 1% /run/user/42
tmpfs tmpfs 1.6G 4.0K 1.6G 1% /run/user/1000
/dev/loop0 squashfs 2.5G 2.5G 0 100% /usr/src/KERNELS
x8:/tmp>
```

Figure 19.1: Using df

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 19.2 Disk Usage

# du: Disk Usage

- du (**disk usage**) is used to look at both disk capacity and usage
- Display disk usage for the current directory:  
  \$ du
- List all files, not just directories:  
  \$ du -a
- List in human-readable format:  
  \$ du -h
- Display disk usage for a specific directory:  
  \$ du -h somedir

Try the following command:

```
$ find . -maxdepth 1 -type d -exec du -shx {} \; | sort -hr
```

```
c8:/home/coop/.cache>find . -maxdepth 1 -type d -exec du -shx {} \; | sort -hr
338M .
229M ./google-chrome
86M ./spotify
7.7M ./tracker
6.8M ./thunderbird
4.9M ./gnome-software
2.5M ./mesa_shader_cache
956K ./gstreamer-1.0
436K ./samba
52K ./evolution
12K ./fontconfig
8.0K ./vmware
8.0K ./Slack
8.0K ./skypeforlinux
4.0K ./libgweather
4.0K ./googleearth_CACHE
4.0K ./gnome-shell
4.0K ./gnome-screenshot
c8:/home/coop/.cache>
```

Figure 19.2: Using du

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 19.3 Swap

# Using swap

- Virtual memory on disk
- Create swap partition in **fdisk**
- **mkswap**: format swap partitions or files
- **swapon**: activate swap partitions or files
- **swapoff**: deactivate swap partitions or files
- Can mount at boot time (using **/etc/fstab**)
- Can have more than one swap partition

```
$ cat /proc/swaps
```

| Filename     | Type      | Size    | Used | Priority |
|--------------|-----------|---------|------|----------|
| /dev/sda6    | partition | 8290300 | 0    | -1       |
| /tmp/swpfile | file      | 102396  | 0    | -2       |

Linux employs a **virtual memory** system in which the operating system can function as if it had more memory than it really does. This kind of memory **overcommissioning** functions in two ways:

- Many programs do not actually use all the memory they are given permission to use. Sometimes this is because child processes inherit a copy of the parent's memory regions utilizing a **COW** (Copy On Write) technique in which the child only obtains a unique copy (on a page by page basis) when there is a change.
- When memory pressure becomes important less active memory regions may be **swapped** out to disk, to be recalled only when needed again.

Such swapping is usually done to one or more dedicated partitions or files; Linux permits multiple **swap areas** so the needs can be adjusted dynamically. Each area has a priority and lower priority areas are not used until higher priority areas are filled.

In most situations the recommended swap size is the total **RAM** on the system. You can see what your system is currently using for swap areas with `cat /proc/swaps` and report on current usage with `free`. At any given time most memory is in use for caching file contents to prevent actually going to the disk any more than necessary, or in a sub-optimal order or timing. Such pages of memory are never swapped out as the backing store is the files themselves, so writing out to swap would be pointless; instead **dirty** pages (memory containing updated file contents that no longer reflect the stored data) are flushed out to disk.

It is also worth pointing out that in Linux memory used by the kernel itself, as opposed to application memory, is **never swapped** out, in distinction to some other operating systems.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 19.4 Filesystem Quotas \*\*

# Filesystem Quotas \*\*

- **Linux** can use and enforce quotas on many filesystem types.
- These utilities help manage quotas:
  - **quotacheck**: generates and updates quota accounting files
  - **quotaon**: enables quota accounting
  - **quotaoff**: disables quota accounting
  - **edquota**: edits user or group quotas
  - **quota**: reports on usage and limits
- Quota operations require the existence of the files **aquota.user** and **aquota.group** in the root directory of the filesystem using quotas.



### Please Note

The use of disk quotas is no longer popular in enterprise data centers, which is why this section is marked as optional!

Disk quotas allow administrators to control the maximum space particular users (or groups) are allowed. Considerable flexibility is allowed and quotas can be assigned on a per-filesystem basis. Protection is provided against a subset of users exhausting collective resources.

Quotas may be enabled or disabled on a per-filesystem basis. In addition **Linux** supports the use of quotas based on user and group ids.

Different filesystem types may have additional quota-related utilities, such as **xfs\_quota**.

# Setting up quotas

- Mount the filesystem with user and/or group quota options:
  - Add the `usrquota` and/or `grpquota` options to the filesystems entry in `/etc/fstab`
  - Mount or remount the filesystem
- Run **quotacheck** on the filesystem to set up quotas
- Enable quotas on the filesystem
- Set quotas with **edquota**

To create a filesystem quota you must first make sure you have mounted the filesystem with the user and/or group quota mount options. Without these nothing else will work.

For example, to enable use quotas, first you need to put the right options in `/etc/fstab` as in:

```
/dev/sda5 /home ext4 defaults,usrquota 1 2
```

where we have assumed `/home` is on a dedicated partition.

Then test with the following commands:

```
$ sudo mount -o remount /home
$ sudo quotacheck -vu /home
$ sudo quotaon -vu /home
$ sudo edquota someusername
```

You may also want to set up **grace periods** with **edquota**.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## quotacheck

- **quotacheck** creates and updates the filesystem's quota accounting files (`aquota.user` and `aquota.group`)
- To update user files for all filesystems in `/etc/fstab` with user quota options:  
`$ sudo quotacheck -ua`
- To update group files for all filesystems in `/etc/fstab` with group quota options:  
`$ sudo quotacheck -ga`
- To update the user file for a particular filesystem:  
`$ sudo quotacheck -u [somefilesystem]`
- To update the group file for a particular filesystem:  
`$ sudo quotacheck -g [somefilesystem]`

Use the `-v` option to get more verbose output.

**quotacheck** is generally only run when quotas are initially turned on (or need to be updated). The program may also be run when **fsck** reports errors in the filesystem when the system is starting up.

# Turning Quotas On and Off

- **quotaon** and **quotaoff** are used to turn filesystem quotas on and off

```
$ sudo quotaon --help
```

quotaon: Usage:

|                                                         |                                      |
|---------------------------------------------------------|--------------------------------------|
| quotaon [-guvp] [-F quotaformat] [-x state] -a          |                                      |
| quotaon [-guvp] [-F quotaformat] [-x state] filesys ... |                                      |
| -a, --all                                               | turn quotas on for all filesystems   |
| -f, --off                                               | turn quotas off                      |
| -u, --user                                              | operate on user quotas               |
| -g, --group                                             | operate on group quotas              |
| -p, --print-state                                       | print whether quotas are on or off   |
| -x, --xfs-command=cmd                                   | perform XFS quota command            |
| -F, --format=formatname                                 | operate on specific quota format     |
| -v, --verbose                                           | print more messages                  |
| -h, --help                                              | display this help text and exit      |
| -V, --version                                           | display version information and exit |

- Note that these two programs are really one and the same and operate accordingly to which name they are called with.

For example:

```
$ sudo quotaon -av
```

```
1 /dev/sda6 [/]: group quotas turned on
2 /dev/sda5 [/home]: user quotas turned on
```

```
$ sudo quotaoff -av
```

```
1 /dev/sda6 [/]: group quotas turned off
2 /dev/sda5 [/home]: user quotas turned off
```

```
$ sudo quotaon -avu
```

```
1 dev/sda5 [/home]: user quotas turned on
```

```
$ sudo quotaoff -avg
```

```
1 /dev/sda6 [/]: group quotas turned off
```

Note also that quota operations will fail if the files aquota.user or aquota.group do not exist.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Examining Quotas

- The **quota** utility is used to generate reports on quotas:
  - quota (or quota -u) returns your current user quota.
  - quota -g returns your current group quota
  - The superuser may look at quotas for any user or group by specifying a user or group name.

```
$ sudo quota george
1 Disk quotas for user george (uid 1000):
2 Filesystem blocks quota limit grace files quota limit grace
3 /dev/sda5 837572 500 1000 5804 0 0
```

```
$ sudo quota gracie
1 Disk quotas for user gracie (uid 1001):
2 Filesystem blocks quota limit grace files quota limit grace
3 /dev/sda5 83757 5000 10000 5804 0 0
```

# Setting Quotas

- Edit limits for `username` :

```
$ sudo edquota -u [username]
```

- Edit limits for `groupname` :

```
$ sudo edquota -g [groupname]
```

- Copy `userproto`'s user quota values to `username` :

```
$ sudo edquota -u -p [userproto] [username]
```

- Copy `groupproto`'s group quota values to `groupname` :

```
$ sudo edquota -g -p [groupproto] [groupname]
```

– The last two commands are useful for including in scripts which might be used to create new accounts and set quotas for them.

- Set grace periods :

```
$ sudo edquota -t
```

Typing **edquota** brings up the quota editor. For the user or group specified a temporary file is created with a text representation of the current disk quotas for that user or group.

Then an editor is invoked for that file and quotas may then be modified. Once you leave the editor, the temporary file is read and the binary quota files adopt the changes.

The only fields which can be edited in the quota are the soft and hard limits. The other fields are informational only.

Quotas for users and groups may be set for disk blocks and/or inodes. In addition, soft and hard limits may be set as well as grace periods: Soft limits may be exceeded for a grace period. Hard limits may never be exceeded.

The grace period is set on a per-filesystem basis.

```
$ sudo edquota gracie
$ sudo edquota -t
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 19.5 Labs



### Video Demonstration Resources

[using\\_swap\\_and\\_oom\\_demo.mp4](#)

## Exercise 19.1: Managing Swap Space

Examine your current swap space by doing:

```
$ cat /proc/swaps
```

| 1 | Filename   | Type      | Size    | Used | Priority |
|---|------------|-----------|---------|------|----------|
| 2 | /dev/sda11 | partition | 4193776 | 0    | -1       |

We will now add more swap space by adding either a new partition or a file. To use a file we can do:

```
$ dd if=/dev/zero of=swpfile bs=1M count=1024
```

```
1 1024+0 records in
2 1024+0 records out
3 1073741824 bytes (1.1 GB) copied, 1.30576 s, 822 MB/s
```

```
$ mkswap swpfile
```

```
1 Setting up swapspace version 1, size = 1048572 KiB
2 no label, UUID=85bb62e5-84b0-4fdd-848b-4f8a289f0c4c
```

(For a real partition just feed **mkswap** the partition name, but be aware all data on it will be erased!)

Activate the new swap space:

```
$ sudo swapon swpfile
```

```
1 swapon: /tmp/swpfile: insecure permissions 0664, 0600 suggested.
2 swapon: /tmp/swpfile: insecure file owner 500, 0 (root) suggested.
```



### On RedHat, CentOS, or Fedora

Notice **RHEL** warns us we are being insecure, we really should fix with:

```
$ sudo chown root:root swpfile
$ sudo chmod 600 swpfile
```

We ensure swpfile is being used:

```
$ cat /proc/swaps
```

| 1 | Filename     | Type      | Size    | Used | Priority |
|---|--------------|-----------|---------|------|----------|
| 2 | /dev/sda11   | partition | 4193776 | 0    | -1       |
| 3 | /tmp/swpfile | file      | 1048572 | 0    | -2       |

Note the Priority field; swap partitions or files of lower priority will not be used until higher priority ones are filled.

Remove the swap file from use and delete it to save space:

For the exclusive use of LFS301 corp class taught 06 to 09 December

```
$ sudo swapoff swpfile
$ sudo rm swpfile
```

## Exercise 19.2: Filesystem Quotas



### Please Note

The subsection describing this material was marked as optional, so you may not have covered the material necessary to do this exercise.

1. Change the entry in `/etc/fstab` for your new filesystem to use user quotas (change `noexec` to `usrquota` in the entry for `/mnt/tempdir`). Then remount the filesystem.
2. Initialize quotas on the new filesystem, and then turn the quota checking system on.
3. Now set some quota limits for the normal user account: a soft limit of 500 blocks and a hard limit of 1000 blocks.
4. As the normal user, attempt to use `dd` to create some files to exceed the quota limits. Create `bigfile1` (200 blocks) and `bigfile2` (400 blocks).  
You should get a warning. Why?
5. Create `bigfile3` (600 blocks).  
You should get an error message. Why? Look closely at the file sizes.
6. Eliminate the persistent mount line you inserted in `/etc/fstab`.

## ✓ Solution 19.2

1. Change `/etc/fstab` to have one of the following two lines according to whether you are using a real partition or a loopback file:



in `/etc/fstab`

```
/dev/sda11 /mnt/tempdir ext4 usrquota 1 2
/imagefile /mnt/tempdir ext4 loop,usrquota 1 2
```

Then remount:

```
$ sudo mount -o remount /mnt/tempdir
```

2. `$ sudo quotacheck -u /mnt/tempdir`  
`$ sudo quotaon -u /mnt/tempdir`  
`$ sudo chown student.student /mnt/tempdir`

(You won't normally do the line above, but we are doing it to make the next part easier).

3. Substitute your user name for the `student` user account.
4. `$ sudo edquota -u student`

5. `$ cd /mnt/tempdir`  
`$ dd if=/dev/zero of=bigfile1 bs=1024 count=200`

1 200+0 records in  
 2 200+0 records out  
 3 204800 bytes (205 kB) copied, 0.000349604 s, 586 MB/s

For the exclusive use of LFS301 corp class taught 06 to 09 December

```
$ quota
```

```
1 Disk quotas for user student (uid 500):
2 Filesystem blocks quota lim grace files qu lim gr
3 /dev/sda11 200 500 1000 1 0 0
```

```
$ dd if=/dev/zero of=bigfile2 bs=1024 count=400
```

```
1 sda11: warning, user block quota exceeded.
2 400+0 records in
3 400+0 records out
4 4096600 bytes (410 kB) copied, 0.000654847 s, 625 MB/s
```

Create `bigfile3` (600 blocks).

6. \$ quota

```
1 Disk quotas for user student (uid 500):
2 Filesystem blocks quota limit grace files quota limit grace
3 /dev/sda11 600* 500 1000 6days 2 0 0
```

```
$ dd if=/dev/zero of=bigfile3 bs=1024 count=600
```

```
1 sda11: write failed, user block limit reached.
2 dd: writing `bigfile3': Disk quota exceeded
3 401+0 records in
4 400+0 records out
5 409600 bytes (410 kB) copied, 0.00177744 s, 230 MB/s
```

```
$ quota
```

```
1 Disk quotas for user student (uid 500):
2 Filesystem blocks quota limit grace files quota limit grace
3 /dev/sda11 1000* 500 1000 6days 3 0 0
```

```
$ ls -l
```

```
1 total 1068
2 -rw----- 1 root root 7168 Dec 10 18:56 aquota.user
3 -rw-rw-r-- 1 student student 204800 Dec 10 18:58 bigfile1
4 -rw-rw-r-- 1 student student 409600 Dec 10 18:58 bigfile2
5 -rw-rw-r-- 1 student student 409600 Dec 10 19:01 bigfile3
6 drwx----- 2 root root 16384 Dec 10 18:47 lost+found
7 -rwxr-xr-x 1 root root 41216 Dec 10 18:52 more
```

Look closely at the file sizes.

7. Get rid of the line in `/etc/fstab`.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 20

# The Ext4 Filesystems



|      |                                                       |     |
|------|-------------------------------------------------------|-----|
| 20.1 | ext4 Features . . . . .                               | 256 |
| 20.2 | ext4 Layout and Superblock and Block Groups . . . . . | 257 |
| 20.3 | dumpe2fs . . . . .                                    | 260 |
| 20.4 | tune2fs . . . . .                                     | 261 |
| 20.5 | Labs . . . . .                                        | 262 |

## 20.1 ext4 Features

# ext4 Filesystem Features

- Backward compatible with **ext3** and **ext2** ancestors
- Uses **extents** instead of **block lists**
- Persistent pre-allocation
- Delayed allocation
- Increases the 32,000 subdirectory limit of earlier ancestors
- Journal checksum
- Fast filesystem checking
- Precise timestamps (nanoseconds)



### Extents

An extent is a group of contiguous blocks. The use of extents can improve large file performance and reduce fragmentation.

The **ext4** filesystem can support volumes up to 1 EB and file sizes up to 16 TB. **Extents** replace the older block mapping mechanism.

**ext4** is backwards compatible with **ext3** and **ext2**. It can pre allocate disk space for a file. The allocated space is usually guaranteed and contiguous. It also uses a performance technique called **allocate-on-flush** (delays block allocation until it writes data to disk). **ext4** breaks the 32,000 subdirectory limit of **ext3**.

**ext4** uses checksums for the journal which improves reliability. This can also safely avoid a disk I/O wait during journalling, which results in a slight performance boost.

Another feature is the use of improved timestamps. **ext4** provides timestamps measured in nanoseconds.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 20.2 ext4 Layout and Superblock and Block Groups

# ext4 Superblock and Block Groups

- **Superblock** at the beginning contains information about the entire filesystem
- It is followed by **Block Groups** composed of sets of contiguous blocks
  - Include administrative information
  - High redundancy of information in block groups
  - Other blocks store file data
- Block size is specified when filesystem is created
  - May be 512, 1K, 2K, 4K, 8K etc. bytes, but not larger than a **page** of memory (4 KB on **x86**).

An **ext4** filesystem is split into a set of block groups. The block allocator tries to keep each file's blocks within the same block group to reduce seek times. The default block size is 4 KB, which would create a block group of 128 MB.

All fields in **ext4** are written to disk in **little-endian** order, except the journal.

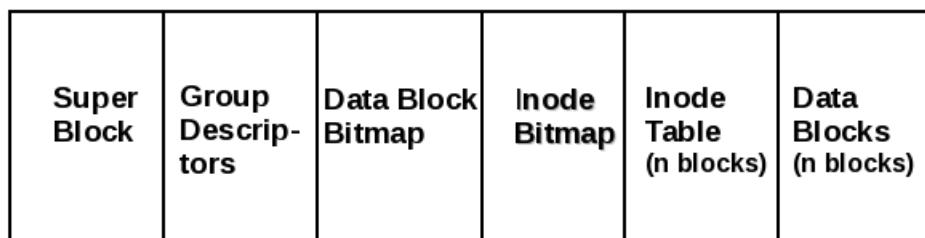


Figure 20.1: **ext[3,4]** Filesystem Layout

The layout of a standard block group is simple. For block group 0, the first 1024 bytes are unused (to allow for boot sectors, etc). The superblock will start at the first block, except for block group 0. This is followed by the group descriptors and a number of **GDT** blocks. These are followed by the data block bitmap, the inode bitmap, the inode table, and the data blocks.

# Block Groups

- Duplicate Superblocks and Group Descriptors help in filesystem recovery

```
student@gentoo:~$ sudo dumpe2fs /dev/sda1 | grep superblock
dumpe2fs 1.43.3 (04-Sep-2016)
 Primary superblock at 0, Group descriptors at 1-3
 Backup superblock at 32768, Group descriptors at 32769-32771
 Backup superblock at 98304, Group descriptors at 98305-98307
 Backup superblock at 163840, Group descriptors at 163841-163843
 Backup superblock at 229376, Group descriptors at 229377-229379
 Backup superblock at 294912, Group descriptors at 294913-294915
 Backup superblock at 819200, Group descriptors at 819201-819203
 Backup superblock at 884736, Group descriptors at 884737-884739
 Backup superblock at 1605632, Group descriptors at 1605633-1605635
 Backup superblock at 2654208, Group descriptors at 2654209-2654211
 Backup superblock at 4096000, Group descriptors at 4096001-4096003
student@gentoo:~$
```

Figure 20.2: Block Groups

The first and second blocks are the same in every block group, and comprise the **Superblock** and the **Group Descriptors**. Under normal circumstances, only those in the first block group are used by the kernel; the duplicate copies are only referenced when the filesystem is being checked. If everything is OK, the kernel merely copies them over from the first block group.

If there is a problem with the master copies, it goes to the next and so on until a healthy one is found and the filesystem structure is rebuilt. This redundancy makes it very difficult to thoroughly fry an ext2/3/4 filesystem, as long as the filesystem checks are run periodically.

In the early incarnations of the ext filesystem family, each block group contained the group descriptors for every block group, as well as a copy of the superblock. As an optimization, however, today not all block groups have a copy of the superblock and group descriptors. To see what you have, you could examine it as in the accompanying screenshot, (putting in an appropriate device node) to see precise locations. This happens when the filesystem is created with the `sparse_super` option, which is the default.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## A Closer Look at the Superblock

- The Superblock contains global information about the filesystem
  - Mount count and maximum mount count
  - Block size for this filesystem
  - Blocks per group
  - Free block count
  - Free Inode count
  - OS ID
- The Superblock is redundantly stored in several block groups

Note that every time the disk is successfully mounted, mount count is incremented. The filesystem is checked every maximum-mount-counts or every 180 days whichever comes first.

Block size can be set through the **mkfs** command.

The superblock for the filesystem is stored in block 0 of the disk. This superblock contains information about the filesystem itself.

## 20.3 dumpe2fs

### Block and Inode Information for ext4: dumpe2fs

- Block size used to set maximum number of:
  - Blocks
  - Inodes
  - Superblocks
- See **dumpe2fs** for filesystem info:
  - Limits
  - Capabilities and flags
  - Other attributes

You can use the **dumpe2fs** program to get information about a particular partition.

```
$ sudo dumpe2fs /dev/sdb1

dumpe2fs 1.45.6 (20-Mar-2020)
Filesystem volume name: VMS
Last mounted on: /VMS
Filesystem UUID: fc621c7-e2ce-414a-8a7e-e2311640802f
Filesystem magic number: 0xEF53
Filesystem revision #: 1 (dynamic)
Filesystem features: has_journal ext_attr resize_inode dir_index filetype needs_recovery extent 64bit flex_bg \
sparse_super large_file huge_file uninit_bg dir_nlink extra_isize
Filesystem flags: signed_directory_hash
Default mount options: user_xattr acl
Filesystem state: clean
Errors behavior: Continue
Filesystem OS type: Linuxff .
Inode count: 14352384
Block count: 57388288
Reserved block count: 2869413
Free blocks: 22270800
Free inodes: 14352217
First block: 0
Block size: 4096

Block bitmap at 1056 (bg #0 + 1056)
Inode bitmap at 1072 (bg #0 + 1072)
Inode table at 1599-2110 (bg #0 + 1599)
415 free blocks, 8192 free inodes, 0 directories, 8192 unused inodes
Free blocks: 33822-33985, 34550-34691, 38803-38911
Free inodes: 8193-16384
Group 2: (Blocks 65536-98303) csum 0xdde9 [INODE_UNINIT, ITABLE_ZEROED]
Block bitmap at 1057 (bg #0 + 1057)
Inode bitmap at 1073 (bg #0 + 1073)
Inode table at 2111-2622 (bg #0 + 2111)
0 free blocks, 8192 free inodes, 0 directories, 8192 unused inodes
Free blocks:
Free inodes: 16385-24576
....
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 20.4 tune2fs

# tune2fs: Change Filesystem Parameters

**tune2fs** can be used to change filesystem parameters.

- To change the maximum number of mounts between filesystem checks (`max-mount-count`):

```
$ sudo tune2fs -c 25 /dev/sda1
```

- To change the time interval between checks (`interval-between-checks`):

```
$ sudo tune2fs -i 10 /dev/sda1
```

- To list the contents of the superblock including the current values of parameters which can be changed:

```
$ sudo tune2fs -l /dev/sda1
```

basically shows the global information from **dumpe2fs**

```
$ sudo tune2fs -l /dev/sdb1

tune2fs 1.45.6 (20-Mar-2020)
Filesystem volume name: VMS
Last mounted on: /
Filesystem UUID: fce521c7-e2ce-414a-8a7e-e2311640802f
Filesystem magic number: 0xEF53
Filesystem revision #: 1 (dynamic)
Filesystem features: has_journal ext_attr resize_inode dir_index filetype needs_recovery extent 64bit flex_bg sparse_super large_file huge_file uninit_bg dir_nlink extra_isize
Filesystem flags: signed_directory_hash
Default mount options: user_xattr acl
Filesystem state: clean
Errors behavior: Continue
Filesystem OS type: Linux
Inode count: 14352384
Block count: 57388288
Reserved block count: 2869413
Free blocks: 22270800
Free inodes: 14352217
First block: 0
Block size: 4096
Fragment size: 4096
.....
Filesystem created: Mon Sep 25 14:14:57 2017
Last mount time: Mon Mar 8 06:05:03 2021
Last write time: Mon Mar 8 06:05:03 2021
Mount count: 2003
Maximum mount count: -1
Last checked: Wed Feb 28 14:24:15 2018
Check interval: 0 (<none>)
Lifetime writes: 14 TB
....
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 20.5 Labs

### Exercise 20.1: Defragmentation

Newcomers to **Linux** are often surprised at the lack of mention of filesystem **defragmentation** tools, since such programs are routinely used in the **Windows** world.

However, native filesystems in **UNIX**-type operating systems, including **Linux**, tend not to suffer serious problems with filesystem fragmentation.

This is primarily because they do not try to cram files onto the innermost disk regions where access times are faster. Instead, they spread free space out throughout the disk, so that when a file has to be created there is a much better chance that a region of free blocks big enough can be found to contain the entire file in either just one or a small number of pieces.

For modern hardware, the concept of innermost disk regions is obscured by the hardware anyway.



#### Don't do this

For **SSDs** defragmentation can actually shorten the lifespan of the storage media due to finite read/erase/write cycles. On smart operating systems defragmentation is turned off by default on **SSD** drives.

Furthermore, the newer **journalling** filesystems (including **ext4**) work with **extents** (large contiguous regions) by design.

However, there does exist a tool for de-fragmenting **ext4** filesystems:

```
$ sudo e4defrag
```

```
1 Usage : e4defrag [-v] file...| directory...| device...
2 : e4defrag -c file...| directory...| device...
```

**e4defrag** is part of the **e2fsprogs** package and should be on all modern **Linux** distributions.

The only two options are:

- **-v**: Be verbose.
- **-c**: Don't actually do anything, just analyze and report.

The argument can be:

- A file
- A directory
- An entire device

Examples:

```
$ sudo e4defrag -c /var/log
```

```
1 <Fragmented files>
2 1. /var/log/lastlog now/best size/ext
 5/1 9 KB
3 2. /var/log/sa/sa24 3/1 80 KB
4 3. /var/log/rhsm/rhsm.log 2/1 142 KB
5 4. /var/log/messages 2/1 4590 KB
6 5. /var/log/Xorg.1.log.old 1/1 36 KB
7
8 Total/best extents 120/112
9 Average size per extent 220 KB
10 Fragmentation score
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

```

11 [0-30 no problem: 31-55 a little bit fragmented: 56- needs defrag]
12 This directory (/var/log) does not need defragmentation.
13 Done.

```

```
$ sudo e4defrag /var/log
```

```

1 ext4 defragmentation for directory(/var/log)
2 [2/152]/var/log/Xorg.2.log: 100% [OK]
3 [3/152]/var/log/Xorg.0.log.old: 100% [OK]
4 [4/152]/var/log/messages-20141019.gz: 100% [OK]
5 [5/152]/var/log/boot.log: 100% [OK]
6 [7/152]/var/log/cups/page_log-20140924.gz: 100% [OK]
7 [8/152]/var/log/cups/access_log-20141019.gz: 100% [OK]
8 [9/152]/var/log/cups/access_log: 100% [OK]
9 [10/152]/var/log/cups/error_log-20141018.gz: 100% [OK]
10 [11/152]/var/log/cups/error_log-20141019.gz: 100% [OK]
11 [12/152]/var/log/cups/access_log-20141018.gz: 100% [OK]
12 [14/152]/var/log/cups/page_log-20141018.gz: 100% [OK]
13 ...
14 [152/152]/var/log/Xorg.1.log.old: 100% [OK]
15
16 Success: [112/152]
17 Failure: [40/152]

```

Try running **e4defrag** on various files, directories, and entire devices, always trying with **-c** first.

You will generally find that **Linux** filesystems only tend to need defragmentation when they get very full, over 90 percent or so, or when they are small and have relatively large files, like when a **boot** partition is used.

## Exercise 20.2: Modifying Filesystem Parameters with **tune2fs**

We are going to fiddle with some properties of a formatted **ext4** filesystem. This does not require unmounting the filesystem first.

In the below you can work with an image file you create as in:

```
$ dd if=/dev/zero of=imagefile bs=1M count=1024
$ mkfs.ext4 imagefile
```

or you can substitute **/dev/sdaX** (using whatever partition the filesystem you want to modify is mounted on) for **imagefile**.

1. Using **dumpe2fs**, obtain information about the filesystem whose properties you want to adjust.

Display:

- The maximum mount count setting (after which a filesystem check will be forced.)
- The Check interval (the amount of time after which a filesystem check is forced)
- The number of blocks reserved, and the total number of blocks

2. Change:

- The maximum mount count to 30.
- The Check interval to three weeks.
- The percentage of blocks reserved to 10 percent.

3. Using **dumpe2fs**, once again obtain information about the filesystem and compare with the original output.

## Solution 20.2

1. \$ dumpe2fs imagefile > dumpe2fs-output-initial

For the exclusive use of LFS301 corp class taught 06 to 09 December

```
1 dumpe2fs 1.42.9 (28-Dec-2013)
```

```
$ grep -i -e "Mount count" -e "Check interval" -e "Block Count" dumpe2fs-output-initial
```

```
1 Block count: 262144
2 Reserved block count: 13107
3 Mount count: 0
4 Maximum mount count: -1
5 Check interval: 0 (<none>)
```

2. \$ tune2fs -c 30 imagefile

```
1 tune2fs 1.42.9 (28-Dec-2013)
2 Setting maximal mount count to 30
```

```
$ tune2fs -i 3w imagefile
```

```
1 tune2fs 1.42.9 (28-Dec-2013)
2 Setting interval between checks to 1814400 seconds
```

```
$ tune2fs -m 10 imagefile
```

```
1 tune2fs 1.42.9 (28-Dec-2013)
2 Setting reserved blocks percentage to 10% (26214 blocks)
```

3. \$ dumpe2fs imagefile > dumpe2fs-output-final

```
1 dumpe2fs 1.42.9 (28-Dec-2013)
```

```
$ grep -i -e "Mount count" -e "Check interval" -e "Block Count" dumpe2fs-output-final
```

```
1 Block count: 262144
2 Reserved block count: 26214
3 Mount count: 0
4 Maximum mount count: 30
5 Check interval: 1814400 (3 weeks)
```

```
$ diff dumpe2fs-output-initial dumpe2fs-output-final
```

```
1 14c14
2 < Reserved block count: 13107
3 ---
4 > Reserved block count: 26214
5 29c29
6 < Last write time: Wed Oct 26 14:26:19 2016
7 ---
8 > Last write time: Wed Oct 26 14:26:20 2016
9 31c31
10 < Maximum mount count: -1
11 ---
12 > Maximum mount count: 30
13 33c33,34
14 < Check interval: 0 (<none>)
15 ---
16 > Check interval: 1814400 (3 weeks)
17 > Next check after: Wed Nov 16 13:26:16 2016
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 21

# The XFS and BTRFS Filesystems \*\*



|      |             |     |
|------|-------------|-----|
| 21.1 | XFS .....   | 266 |
| 21.2 | btrfs ..... | 268 |
| 21.3 | Labs .....  | 269 |



### Please Note

\*\* These sections may be considered in part or in whole as optional. They contain either background reference material, specialized topics, or advanced subjects. The instructor may choose to cover or not cover them depending on classroom experience and time constraints.

## 21.1 XFS

# XFS

- Default filesystem in **RHEL**
- Designed by **SGI** for larger filesystems
  - Up to 16 EB (exabytes) for the filesystem total
  - Up to 8 EB for one file
- Journalling filesystem
  - Meta-data
  - Quota
  - Can be on an external device
- Built for performance
  - Direct DMA I/O
  - Guaranteed rate I/O
  - Adjustable stripe size to match underlying storage
- Supports extended attributes (**xattrs**)

The **XFS** filesystem was designed and created by **Silicon Graphics Inc.** It was engineered to deal with large data sets for **SGI** systems, and effectively handle parallel I/O tasks.

Unlike other **Linux** filesystems, **XFS** can also journal quota information. This leads to decreased recovery time when a quota-enabled filesystem is uncleanly unmounted.

High performance is one of the key design elements of **XFS**. The filesystem has methods for guaranteeing an I/O rate, allowing direct **DMA** I/O and external journal devices. The filesystem can also match stripe sizes with underlying raid or **LVM** devices.

As with other **Linux** filesystems **XFS** supports extended attributes.

## XFS Details

- Online maintenance tools
  - Defragmentation
  - Enlarging
  - Dump/Restore
- Supports quotas
  - Normal quota tools can be used
  - Supports per-directory quotas
    - \* Must be modified with **XFS** quota tools
- Does not directly support snapshots
  - **xfs\_freeze** command
    - \* Not needed if using **LVM** tools
- Backup and restore with **xfsdump** and **xfsrestore**

Maintaining an **XFS** filesystem is made easier by the fact that most of the maintenance tasks can be done while the filesystem is mounted, or “on-line”. This includes, defragmentation, resizing (enlarge only), and Dump/Restore.

The **xfsdump** and **xfsrestore** commands can be paused and resumed at a later time. They are also multi-threaded, making **XFS** dumps and restores happen much faster.

The traditional quota commands can be used to manipulate per-filesystem quotas on an **XFS** volume. However, if you use the **xfs\_quota** command you can use the per-directory quotas that **XFS** supports.

The **XFS** filesystem does not directly support hardware or software snapshots. However, the utility **xfs\_freeze** can be used to quiesce the filesystem to allow for a snapshot tool to work on the underlying device. The **Linux LVM** tools will automatically use **xfs\_freeze** to quiesce the filesystem for snapshots.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 21.2 btrfs

# btrfs

- **B-TRee File System**
- Copy-on-write (**COW**)
- Snapshots of entire filesystems or subvolumes
- Built in Logical Volume Management
- Max file size: 16 EB
- Max number of files:  $2^{64}$
- Default root filesystem on the **SLES** and **openSUSE** distributions



### Bye Bye btrfs on RHEL

**btrfs** was removed in **RHEL/CentOS 8**. While it can be installed from upstream sources, it is not recommended to use an important filesystem, if it is unsupported by your distribution.

Both **Linux** developers and **Linux** users with high performance and high capacity or specialized needs are following the development and gradual deployment of the **btrfs** filesystem, which was created by Chris Mason. The name stands for **B-tree file system**. Full documentation can be found at [https://btrfs.wiki.kernel.org/index.php/Main\\_Page](https://btrfs.wiki.kernel.org/index.php/Main_Page).

**btrfs** is intended to address the lack of pooling, snapshots, checksums, and integral multi-device spanning in other **Linux** filesystems such as **ext4**. Such features can be crucial for **Linux** to scale into larger enterprise storage configurations.

While **btrfs** has been in the mainline kernel since 2.6.29 it was long viewed as experimental, although vendors have used it in new products, and is now the default in some distributions.

One of the main features is the ability to take frequent **snapshots** of entire filesystems, or sub-volumes of entire filesystems in virtually no time. Because **btrfs** makes extensive use of **COW** techniques (Copy on Write), such a snapshot does not involve any more initial space for data blocks or any I/O activity except for some metadata updating.

One can easily revert to the state described by earlier snapshots and even induce the kernel to reboot off an earlier root filesystem snapshot.

**btrfs** maintains its own internal framework for adding or removing new partitions and/or physical media to existing filesystems, much as **LVM** (Logical Volume Management) does.

Before **btrfs** becomes suitable for day-to-day use in critical filesystems, some tasks require finishing. For a review of the development history of **btrfs** and expected evolution see <https://lwn.net/Articles/575841>.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 21.3 Labs

### Exercise 21.1: Finding Out More About xfs



#### Please Note

We do not have a detailed lab exercise you can do with **xfs**; many systems still will not have the kernel modules and relevant user utilities installed. However, if your **Linux** kernel and distribution does support it, you can easily create a filesystem with `mkfs -t xfs`.

You can find out about available **xfs**-related utilities with:

```
$ man -k xfs
```

|                      |                                                             |
|----------------------|-------------------------------------------------------------|
| 1 attr (1)           | - extended attributes on XFS filesystem objects             |
| 2 bcc-xfsdist (8)    | - Summarize XFS operation latency. Uses Linux eBPF/bcc.     |
| 3 bcc-xfsslower (8)  | - Trace slow xfs file operations, with per-event details.   |
| 4 filesystems (5)    | - Linux filesystem types: ext, ext2, ext3, ext4, hpfs, i... |
| 5 fs (5)             | - Linux filesystem types: ext, ext2, ext3, ext4, hpfs, i... |
| 6 fsck.xfs (8)       | - do nothing, successfully                                  |
| 7 fsfreeze (8)       | - suspend access to a filesystem (Ext3/4, ReiserFS, JFS,... |
| 8 mkfs.xfs (8)       | - construct an XFS filesystem                               |
| 9 xfs (5)            | - layout, mount options, and supported file attributes f... |
| 10 xfs_admin (8)     | - change parameters of an XFS filesystem                    |
| 11 xfs_bmap (8)      | - print block mapping for an XFS file                       |
| 12 xfs_copy (8)      | - copy the contents of an XFS filesystem                    |
| 13 xfs_db (8)        | - debug an XFS filesystem                                   |
| 14 xfs_estimate (8)  | - estimate the space that an XFS filesystem will take       |
| 15 xfs_freeze (8)    | - suspend access to an XFS filesystem                       |
| 16 xfs_fsr (8)       | - filesystem reorganizer for XFS                            |
| 17 xfs_growfs (8)    | - expand an XFS filesystem                                  |
| 18 xfs_info (8)      | - display XFS filesystem geometry information               |
| 19 xfs_io (8)        | - debug the I/O path of an XFS filesystem                   |
| 20 xfs_logprint (8)  | - print the log of an XFS filesystem                        |
| 21 xfs_mdrestore (8) | - restores an XFS metadump image to a filesystem image      |
| 22 xfs_metadump (8)  | - copy XFS filesystem metadata to a file                    |
| 23 xfs_mkfile (8)    | - create an XFS file                                        |
| 24 xfs_ncheck (8)    | - generate pathnames from i-numbers for XFS                 |
| 25 xfs_quota (8)     | - manage use of quota on XFS filesystems                    |
| 26 xfs_repair (8)    | - repair an XFS filesystem                                  |
| 27 xfs_rtcp (8)      | - XFS realtime copy command                                 |
| 28 xfs_spaceman (8)  | - show free space information about an XFS filesystem       |
| 29 xfsdist (8)       | - Summarize XFS operation latency. Uses bpftrace/eBPF.      |
| 30 xfsdump (8)       | - XFS filesystem incremental dump utility                   |
| 31 xfsinvutil (8)    | - xfsdump inventory database checking and pruning utility   |
| 32 xfsrestore (8)    | - XFS filesystem incremental restore utility                |
| 33 xqmstats (8)      | - Display XFS quota manager statistics from /proc           |

Read about these utility programs and see if you can play with them on the filesystem you created.

### Exercise 21.2: Finding Out More About btrfs



#### No btrfs on RHEL/CentOS 8

You cannot do this exercise on any system that does not support **btrfs**, such as **RHEL 8**.

For the exclusive use of LFS301 corp class taught 06 to 09 December

**Please Note**

We do not have a detailed lab exercise you can do with **btrfs**; many systems still will not have the kernel modules and relevant user utilities installed. However, if your **Linux** kernel and distribution support it, you can easily create a filesystem with `mkfs -t btrfs`.

You can find out about available **btrfs**-related utilities with either just typing **btrfs**:

```
$ btrfs
1 usage: btrfs [--help] [--version] <group> [<group>...] <command> [<args>]
2
3 Command groups:
4 subvolume manage subvolumes: create, delete, list, etc
5 filesystem overall filesystem tasks and information
6 balance balance data across devices, or change block groups using filters
7 device manage and query devices in the filesystem
8 scrub verify checksums of data and metadata
9 rescue toolbox for specific rescue operations
10 inspect-internal query various internal information
11 property modify properties of filesystem objects
12 quota manage filesystem quota settings
13 qgroup manage quota groups
14 replace replace a device in the filesystem
15
16 Commands:
17 check Check structural integrity of a filesystem (unmounted).
18 restore Try to restore files from a damaged filesystem (unmounted)
19 send Send the subvolume(s) to stdout.
20 receive Receive subvolumes from a stream
21 help Display help information
22 version Display btrfs-progs version
```

You can get quite a bit more detail by typing:

```
$ btrfs --help
```

Read about these utility programs and see if you can play with them on the filesystem you created.

The command

```
$ man -k btrfs
```

also generates a lot of information:

```
1 btrfs-image (8) - create/restore an image of the filesystem
2 btrfs-show (8) - scan the /dev directory for btrfs partitions and print re...
3 btrfsck (8) - check a btrfs filesystem
4 btrfsctl (8) - control a btrfs filesystem
5 mkfs.btrfs (8) - create an btrfs filesystem
6 btrfs (5) - topics about the BTRFS filesystem (mount options, support...
7 btrfs (8) - a toolbox to manage btrfs filesystems
8 btrfs-balance (8) - balance block groups on a btrfs filesystem
9
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 22

# Encrypting Disks



|      |                                        |     |
|------|----------------------------------------|-----|
| 22.1 | Filesystem Encryption . . . . .        | 272 |
| 22.2 | LUKS . . . . .                         | 274 |
| 22.3 | cryptsetup . . . . .                   | 275 |
| 22.4 | Using an Encrypted Partition . . . . . | 276 |
| 22.5 | Mounting at Boot . . . . .             | 277 |
| 22.6 | Labs . . . . .                         | 278 |

## 22.1 Filesystem Encryption

# Filesystem Encryption

- Protect filesystems from prying eyes
- Protect filesystems from attempts to corrupt the data they contain
- Encryption can be chosen at installation or incorporated later
- **Linux** distributions most often use the **LUKS** method and perform encryption-related tasks using **cryptsetup**

By the end of this section, you should be able to:

- Provide sound reasons for using encryption and know when it is called for.
- Understand how **LUKS** operates through the use of **cryptsetup**.
- Be able to set up and use encrypted filesystems and partitions.
- Know how to configure the system to mount encrypted partitions at boot.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Why Use Encryption?

- Secure sensitive data both for storage and transmission across networks.
- Block device level encryption provides one of the strongest protections against data loss or compromise.
- **Linux** distributions offer the choice of encrypting all or some of your disk partitions during installation.
- One can encrypt partitions post-installation, but encrypting an already existing partition in place requires copying the data over

Encryption should be used wherever sensitive data is being stored and transmitted. Configuring and using block device level encryption provides one of the strongest protections against harm caused by loss or compromise of data contained in hard drives and other media.

Modern **Linux** distributions offer the choice of encrypting all or some of your disk partitions during installation. It is also straightforward to create and format encrypted partitions at a later time, but you can not encrypt an already existing partition in place without a data copying operation.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 22.2 LUKS

# LUKS

- Block device level encryption mainly through the use of **LUKS** (Linux Unified Key Setup).
- **LUKS** is installed on top of **cryptsetup**
- **cryptsetup** can also use:
  - **plain dm-crypt** volumes
  - **loop-AES**
  - **TrueCrypt**-compatible format
- However, **LUKS** is the method most often used in **Linux**.
- Easy to migrate partitions to other disks or systems.
- Can be used to transparently encrypt **swap** partitions.

Modern **Linux** distributions provide block device level encryption mainly through the use of **LUKS** (Linux Unified Key Setup). Using block device encryption is highly recommended for portable systems such as laptops, tablets, and smart phones.

**LUKS** is installed on top of **cryptsetup**, a powerful utility that can also use other methods such as **plain dm-crypt** volumes, **loop-AES** and **TrueCrypt**-compatible format. We will not discuss these alternatives, as **LUKS** (which was originally designed for **Linux** but has also been exported to other operating systems) is the standard method most often used in **Linux**.

The **dm-crypt** kernel module uses the **device mapper** kernel infrastructure that is also heavily used by **LVM**, which we discussed previously.

Because **LUKS** stores all necessary information in the partition header itself, it is rather easy to migrate partitions to other disks or systems.

**LUKS** can also be used to transparently encrypt swap partitions.

## 22.3 cryptsetup

# cryptsetup

- Almost everything done multi-faceted **cryptsetup** utility
- Previously setup and formatted encrypted volumes are mounted and unmounted with normal disk utilities
- Used as:  
`cryptsetup [OPTION...] <action> <action-specific>`
- See possibilities with:  
`$ cryptsetup --help`

Basically, everything is done with the Swiss army knife program **cryptsetup**. Once encrypted volumes have been set up they can be mounted and unmounted with normal utilities.

The general form of a command is:

```
cryptsetup [OPTION...] <action> <action-specific>
```

and a rather full listing of possibilities can be generated by:

```
$ cryptsetup --help
```

```
cryptsetup 2.3.3
Usage: cryptsetup [OPTION...] <action> <action-specific>
 -v, --verbose Shows more detailed error messages
 --debug Show debug messages
 --debug-json Show debug messages including JSON
 metadata
 -c, --cipher=STRING The cipher used to encrypt the disk
 (see /proc/crypto)
 -h, --hash=STRING The hash used to create the encryption
 key from the passphrase
 -y, --verify-passphrase Verifies the passphrase by asking for
 it twice
 -d, --key-file=STRING Read the key from a file
 --master-key-file=STRING Read the volume (master) key from file.
 --dump-master-key Dump volume (master) key instead of
 keyslots info
 -s, --key-size=BITS The size of the encryption key
....
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 22.4 Using an Encrypted Partition

# Using an Encrypted Partition

- Provide previously created partition (`/dev/sdc12`) to **LUKS**:  
`$ sudo cryptsetup luksFormat /dev/sdc12`
- If your kernel does not support the default encryption method used by **cryptsetup**, examine `/proc/crypto` and use a supported method:  
`$ sudo cryptsetup luksFormat --cipher aes /dev/sdc12`
- Make encrypted volume available:  
`$ sudo cryptsetup --verbose luksOpen /dev/sdc12 SECRET`
- Format, mount, use and unmount the partition as in:  
`$ sudo mkfs.ext4 /dev/mapper/SECRET  
$ sudo mount /dev/mapper/SECRET /mnt  
....  
$ sudo umount /mnt`
- Remove the association for now:  
`$ sudo cryptsetup --verbose luksClose SECRET`

If the partition `/dev/sdc12` already exists, the following commands will set up encryption, make it available to **LUKS**, format it, mount it, use it, and unmount it. First we need to give the partition to **LUKS**:

```
$ sudo cryptsetup luksFormat /dev/sdc12
```

You will be prompted for a passphrase that will need to open use of the encrypted volume later. Note you only have to do this step once, when setting up encryption. Your kernel may not support **cryptsetup**'s default encryption method. In that case you can examine `/proc/crypto` to see the methods your system supports, and then you can supply a method as in:

```
$ sudo cryptsetup luksFormat --cipher aes /dev/sdc12
```

You can make the volume available any time with:

```
$ sudo cryptsetup --verbose luksOpen /dev/sdc12 SECRET
```

where you will be prompted to supply the passphrase. You can format and then mount the partition:

```
$ sudo mkfs.ext4 /dev/mapper/SECRET

$ sudo mount /dev/mapper/SECRET /mnt
```

and then use to your heart's content just as if it were an unencrypted partition. When you are done, unmount and then remove the association with:

```
$ sudo umount /mnt

$ sudo cryptsetup --verbose luksClose SECRET
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 22.5 Mounting at Boot

# Mounting at Boot

- Can mount an encrypted partition at boot if there are appropriate entries in:
  - `/etc/fstab`
  - `/etc/crypttab`
- The entry in `/etc/crypttab` can be as simple as:  
`SECRET /dev/sdc12`
- Can also specify the password (a **bad idea**) in this entry and other options.
- The entry in `/etc/fstab` can be as simple as:  
`/dev/mapper/SECRET /mnt ext4 defaults 0 0`

To mount an encrypted partition at boot two conditions have to be satisfied:

1. Make an appropriate entry in `/etc/fstab`. There is nothing special about this and it does not refer to encryption in any way.
2. Add an entry to `/etc/crypttab`. This can be as simple as:

`SECRET /dev/sdc12`



### Don't do this

You can do more in this file, such as specifying the password if you do not want to be prompted at boot (which seems counterproductive security-wise). See `man crypttab` to find out what you can do with this file.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 22.6 Labs



### Video Demonstration Resources

[using\\_encrypted\\_loopback\\_demo.mp4](#)

## Exercise 22.1: Disk Encryption

In this exercise, you will encrypt a partition on the disk in order to provide a measure of security in the event that the hard drive or laptop is stolen. Reviewing the **cryptsetup** documentation first would be a good idea (`man cryptsetup` and `cryptsetup --help`).

1. Create a new partition for the encrypted block device with **fdisk**. Make sure the kernel is aware of the new partition table. A reboot will do this but there are other methods.
2. Format the partition with **cryptsetup** using **LUKS** for the crypto layer.
3. Create the un-encrypted pass through device by opening the encrypted block device, i.e., `secret-disk`.
4. Add an entry to `/etc/crypttab` so that the system prompts for the passphrase on reboot.
5. Format the filesystem as an **ext4** filesystem.
6. Create a mount point for the new filesystem, i.e. `/secret`.
7. Add an entry to `/etc/fstab` so that the filesystem is mounted on boot.
8. Try and mount the encrypted filesystem.
9. Validate the entire configuration by rebooting.

## Solution 22.1

1. `$ sudo fdisk /dev/sda`

Create a new partition (in the below `/dev/sda4` to be concrete) and then either issue:

```
$ sudo partprobe -s
```

to have the system re-read the modified partition table, or reboot (which is far safer).

**Note:** If you can't use a real partition, use the technique in the previous chapter to use a loop device or image file for the same purpose.

2. `$ sudo cryptsetup luksFormat /dev/sda4`
3. `$ sudo cryptsetup luksOpen /dev/sda4 secret-disk`
4. Add the following to `/etc/crypttab`:



in `/etc/crypttab`

```
secret-disk /dev/sda4
```

5. `$ sudo mkfs -t ext4 /dev/mapper/secret-disk`
6. `$ sudo mkdir -p /secret`
7. Add the following to `/etc/fstab`:

For the exclusive use of LFS301 corp class taught 06 to 09 December



## in /etc/fstab

```
/dev/mapper/secret-disk /secret ext4 defaults 1 2
```

8. Mount just the one filesystem:

```
$ sudo mount /secret
```

or mount all filesystems mentioned in [/etc/fstab](#):

```
$ sudo mount -a
```

9. Reboot.

## Exercise 22.2: Encrypted Swap

In this exercise, we will be encrypting the **swap partition**. Data written to the swap device can contain sensitive information. Because swap is backed by an actual partition, it is important to consider the security implications of having an unencrypted swap partition.

The process for encrypting is similar to the previous exercise, except we will not create a file system on the encrypted block device.

In this case, we are also going to use the existing swap device by first de-activating it and then formatting it for use as an encrypted swap device. It would be a little bit safer to use a fresh partition below, or you can safely reuse the encrypted partition you set up in the previous exercise. At the end we explain what to do if you have problems restoring.

You may want to revert back to the original unencrypted partition when we are done by just running **mkswap** on it again when it is not being used, as well as reverting the changes in the configuration files, [/etc/crypttab](#) and [/etc/fstab](#).

1. Find out what partition you are currently using for swap and then deactivate it:

```
$ cat /proc/swaps
```

| 1 | Filename   | Type      | Size    | Used | Priority |
|---|------------|-----------|---------|------|----------|
| 2 | /dev/sda11 | partition | 4193776 | 0    | -1       |

```
$ sudo swapoff /dev/sda11
```

2. Do the same steps as in the previous exercise to set up encryption:

```
$ sudo cryptsetup luksFormat /dev/sda11 # may use --cipher aes option
$ sudo cryptsetup luksOpen /dev/sda11 swapcrypt
```

3. Format the encrypted device to use with swap:

```
$ sudo mkswap /dev/mapper/swapcrypt
```

4. Now test to see if it actually works by activating it:

```
$ sudo swapon /dev/mapper/swapcrypt
$ cat /proc/swaps
```

5. To ensure the encrypted swap partition can be activated at boot you need to do two things:

- (a) Add a line to [/etc/crypttab](#) so that the system prompts for the passphrase on reboot:



## in /etc/crypttab

```
swapcrypt /dev/sda11 /dev/urandom swap,cipher=aes-cbc-essiv:sha256,size=256
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

(Note `/dev/urandom` is preferred over `/dev/random` for reasons involving potential **entropy shortages** as discussed in the `man` page for `crypttab`.) You don't need the detailed options that follow, but we give them as an example of what more you can do.

- (b) Add an entry to `/etc/fstab` so that the swap device is activated on boot.



in `/etc/fstab`

```
/dev/mapper/swapcrypt none swap defaults 0 0
```

6. You can validate the entire configuration by rebooting.

To restore your original unencrypted partition:

```
$ sudo swapoff /dev/mapper/swapcrypt
$ sudo cryptsetup luksClose swapcrypt
$ sudo mkswap /dev/sda11
$ sudo swapon -a
```

If the `swapon` command fails it is likely because `/etc/fstab` no longer properly describes the swap partition. If this partition is described in there by actual device node (`/dev/sda11`) there won't be a problem. You can fix either by changing the line in there to be:



in `/etc/fstab`

```
/dev/sda11 swap swap defaults 0 0
```

or by giving a label when formatting and using it as in:

```
$ sudo mkswap -L SWAP /dev/sda11
```

and then putting in the file:



in `/etc/fstab`

```
LABEL=SWAP swap swap defaults 0 0
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 23

# Logical Volume Management (LVM)



|      |                                           |     |
|------|-------------------------------------------|-----|
| 23.1 | Logical Volume Management (LVM) . . . . . | 282 |
| 23.2 | Volumes and Volume Groups . . . . .       | 283 |
| 23.3 | Working with Logical Volumes . . . . .    | 284 |
| 23.4 | Resizing Logical Volumes . . . . .        | 287 |
| 23.5 | LVM Snapshots ** . . . . .                | 288 |
| 23.6 | Labs . . . . .                            | 289 |



### Please Note

\*\* These sections may be considered in part or in whole as optional. They contain either background reference material, specialized topics, or advanced subjects. The instructor may choose to cover or not cover them depending on classroom experience and time constraints.

## 23.1 Logical Volume Management (LVM)

# LVM

- Multiple disks grouped together to create volumes
- Similar features as RAID devices
  - Can actually be built on top of RAID devices
- Better scalability than RAID
  - Can easily be resized (up or down)
  - Can add more devices to grow

**LVM** (Logical Volume Management) breaks up one virtual partition into multiple chunks each of which can be on different partitions and/or disks.

There are many advantages to using **LVM**; in particular it becomes really easy to change the size of the logical partitions and filesystems, to add more storage space, rearrange things etc.

One or more **physical volumes** (disk partitions) are grouped together into a **volume group**. Then the volume group is subdivided into **logical volumes** which mimic the system nominal physical disk partitions and can be formatted to contain mountable filesystems.

There are a variety of command line utilities tasked to create, delete, resize, etc. physical and logical volumes. For a graphical tool most **Linux** distributions use **system-config-lvm**. However, **RHEL** no longer supports this tool and there is currently no graphical tool that works reliably with the most recent variations in filesystem types etc. Fortunately, the command line utilities are not hard to use and are more flexible anyway.

**LVM** does impact performance. There is a definite additional cost that comes from the overhead of the **LVM** layer. However, even on non-**RAID** systems, if you use **striping** (splitting of data to more than one disk) you can achieve some parallelization improvements.

Logical volumes have features similar to **RAID** devices (which we will discuss next.) They can actually be built on top of a **RAID** device. This would give the logical volume the redundancy of a **RAID** device with the scalability of **LVM**.

**LVM** has better scalability than **RAID**: logical volumes can easily be resized; i.e., enlarged or shrunk, as needs require. If more space is needed, additional devices can be added to the logical volume at any time.

## 23.2 Volumes and Volume Groups

# Volumes and Volume Groups

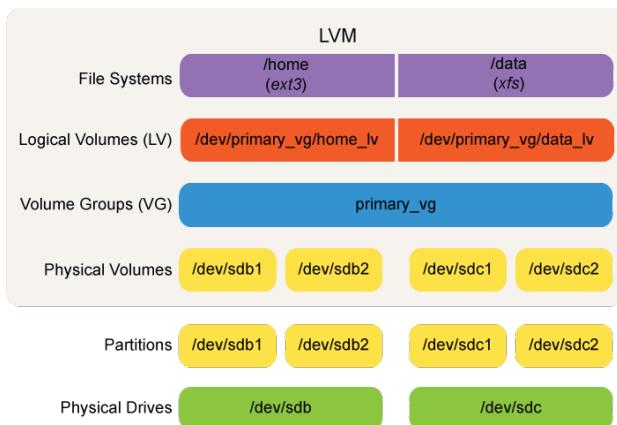


Figure 23.1: LVM components

- Devices are converted to physical volumes

- Multiple physical volumes are grouped into volume groups
  - Volume groups divided into extents
  - Extent size can be defined (4 MB default)
- Logical volumes allocated from volume groups
  - Can be defined by size or number of extents
  - Filesystems are built on logical volumes
  - Can be named anything

Partitions are converted to physical volumes and multiple physical volumes are grouped into volume groups; there can be more than one volume group on the system. Space in the volume group is divided into **extents**; these are 4 MB in size by default, but the size can easily be changed when being allocated.

There are a number of command line utilities used to create and manipulate volume groups, whose name always start with **vg** including:

- **vgcreate**: Creates volume groups.
- **vgextend**: Adds to volume groups.
- **vgreduce**: Shrinks volume groups.

Utilities that manipulate what physical partitions enter or leave volume groups start with **pv** and include:

- **pvcreate**: Converts a partition to a physical volume.
- **pvdisplay**: Shows the physical volumes being used.
- **pvmove**: Moves the data from one physical volume within the volume group to others; this might be required if a disk or partition is being removed for some reason. It would then be followed by:
- **pvremove**: Remove a partition from a physical volume.

Typing `man lvm` will give a full list of **LVM** utilities.

For the exclusive use of LFS301 corp class taught 06 to 09 December

### 23.3 Working with Logical Volumes

## Logical Volume Utilities

```
student@ubuntu:~$ ls -lF /sbin/lv*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/lvchange -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/lvconvert -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/lvcreate -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/lvdisplay -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/lvextend -> lvm*
-rwxr-xr-x 1 root root 2862872 Feb 13 2020 /sbin/lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/lvmconfig -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/lvmdiskscan -> lvm*
-rwxr-xr-x 1 root root 10312 Feb 13 2020 /sbin/lvmdump*
-rwxr-xr-x 1 root root 237624 Feb 13 2020 /sbin/lvmpolld*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/lvmsadc -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/lvmsar -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/lvreduce -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/lvremove -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/lvrename -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/lvresize -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/lvs -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/lvscan -> lvm*
student@ubuntu:~$
```

Figure 23.2: Logical Volume Utilities

There are a number of utilities that manipulate logical volumes. Unsurprisingly, they all begin with **lv**. We will discuss the most commonly used ones, but a short list can be obtained by:

```
$ ls -lF /sbin/lv*
```

These utilities are in `/sbin` not `/usr/sbin` as they may be needed either for boot or repair and recovery.

Most of them are symbolically linked to **lvm**, a Swiss army knife program that does all the work but figures out what it is being asked to do based on the name it is invoked with. This is also true for most of the **pv\*** and **vg\*** utilities as you can verify easily enough.

```
student@ubuntu:~$ ls -lF /sbin/pv*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/pvchange -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/pvck -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/pvcreate -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/pvdisplay -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/pvmove -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/pvremove -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/pvresize -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/pvs -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/pvscan -> lvm*
student@ubuntu:~$
```

Figure 23.3: Physical Volume Utilities

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Creating Logical Volumes

1. Create partitions on disk drives (type 8e in **fdisk**) .
2. Create physical volumes from the partitions.
3. Create the volume group.
4. Allocate logical volumes from the volume group.
5. Format the logical volumes.
6. Mount the logical volumes (also update **/etc/fstab** as needed).

**lvcreate** allocates logical volumes from within volume groups. The size can be specified either in bytes or number of extents (remember they are 4 MB by default). Names can be anything desired.

**lvdiskdisplay** reports on available logical volumes

Filesystems are placed in logical volumes and are formatted with **mkfs** as usual.

For example, assuming one has already created partitions **/dev/sdb1** and **/dev/sdc1** and given them type 8e, the steps involved in setting up and using a new logical volume are:

```
$ sudo pvcreate /dev/sdb1
$ sudo pvcreate /dev/sdc1
$ sudo vgcreate -s 16M vg /dev/sdb1
$ sudo vgextend vg /dev/sdc1
$ sudo lvcreate -L 50G -n mylvm vg
$ sudo mkfs -t ext4 /dev/vg/mylvm
$ mkdir /mylvm
$ sudo mount /dev/vg/mylvm /mylvm
```

Be sure to add the line

```
/dev/vg/mylvm /mylvm ext4 defaults 1 2
```

to **/etc/fstab** to make this a persistent mount.

# Displaying Logical Volumes

- **pvdisplay** - show physical volumes

```
$ pvdisplay
$ pvdisplay /dev/sda5
```

- **vgdisplay** - show volume groups

```
$ vgdisplay
$ vgdisplay /dev/vg0
```

- **lvdisplay** - show logical volumes

```
$ lvdisplay
$ lvdisplay /dev/vg0/lvm1
```

**pvdisplay** is for showing one or more physical volumes. If you leave off physical volume name, it lists all physical volumes.

**vgdisplay** is for showing one or more volume groups. If you leave off the volume group name, it lists all volume groups.

**lvdisplay** is for showing one or more logical volumes. If you leave off the logical volume name, it lists all logical volumes.

```
student@ubuntu:~$ ls -lF /sbin/vg*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/vgcfgbackup -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/vgcfgrestore -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/vgchange -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/vgck -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/vgconvert -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/vgcreate -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/vgdisplay -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/vgexport -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/vgextend -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/vgimport -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/vgimportclone -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/vgmerge -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/vgmknodes -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/vgreduce -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/vgremove -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/vgrename -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/vgs -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/vgscan -> lvm*
lrwxrwxrwx 1 root root 3 Feb 13 2020 /sbin/vgsplit -> lvm*
student@ubuntu:~$
```

Figure 23.4: Volume Group Utilities

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 23.4 Resizing Logical Volumes

# Resizing Logical Volumes

- Much easier than resizing a physical partition
- Extents can come from anywhere in the volume group
- Filesystem can be shrunk or extended while still online
- Can change size either by number of extents or size in bytes (M,G, etc)
- Best done with **lvresize** as in:  
`$ sudo lvresize -r -L 20 GB /dev/VG/mylvm`  
where the **-r** option causes resizing the filesystem at same time as changing volume size
- Can also use **lvextend**, **lvreduce** with **resize2fs**
- Resizing utilities are filesystem-specific

To grow a logical volume with an **ext4** filesystem:

```
$ sudo lvresize -r -L +100M /dev/vg/mylvm
```

where the plus sign indicates adding space. Note that one need not unmount the filesystem to grow it.

To shrink the filesystem:

```
$ sudo lvresize -r -L 200M /dev/vg/mylvm
```

You can also reduce a volume group as in:

```
$ sudo pvmove /dev/sdc1
$ sudo vgreduce vg /dev/sdc1
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 23.5 LVM Snapshots \*\*

# LVM Snapshots

- Exact copy of an existing logical volume
- Useful for backups, application testing, and VMs
- Original state of snapshot is block map
- Only uses space for storing deltas
  - When original logical volume changed, original data blocks copied to snapshot
  - If data added to snapshot, stored only there

**LVM Snapshots** create an exact copy of an existing logical volume.

They are useful for backups, application testing, and deploying **VMs** (Virtual Machines). The original state of the snapshot is kept as the block map.

Snapshots only use space for storing deltas:

- When the original logical volume changes, original data blocks are copied to the snapshot.
- If data is added to snapshot, it is stored only there.

To create a snapshot of an existing logical volume:

```
$ sudo lvcreate -l 128 -s -n mysnap /dev/vg/mylvm
```

To then make a mount point and mount the snapshot:

```
$ mkdir /mysnap
$ mount -o ro /dev/vg/mysnap /mysnap
```

To use the snapshot and to remove the snapshot:

```
$ sudo umount /mysnap
$ sudo lvremove /dev/vg/mysnap
```

Always be sure to remove the snapshot when you are through with it. If you do not remove the snapshot and it fills up because of changes, it will be automatically disabled. A snapshot with the size of the original will never overflow.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 23.6 Labs



### Video Demonstration Resources

[using\\_lvm\\_demo.mp4](#)

#### Exercise 23.1: Logical Volumes

We are going to create a logical volume using two 250 MB partitions. We are going to assume you have real partitionable disk space available.

1. Create two 250 MB partitions of type logical volume (8e).
2. Convert the partitions to physical volumes.
3. Create a volume group named `myvg` and add the two physical volumes to it. Use the default extent size.
4. Allocate a 300 MB logical volume named `mylvm` from volume group `myvg`.
5. Format and mount the logical volume `mylvm` at `/mylvm`
6. Use `lvdisplay` to view information about the logical volume.
7. Grow the logical volume and corresponding filesystem to 350 MB.

#### Solution 23.1

1. Execute:

```
$ sudo fdisk /dev/sda
```

using whatever hard disk is appropriate, and create the two partitions. While in `fdisk`, typing `t` will let you set the partition type to 8e. While it doesn't matter if you don't set the type, it is a good idea to lessen confusion. Use `w` to rewrite the partition table and exit, and then

```
$ sudo partprobe -s
```

or reboot to make sure the new partitions take effect.

2. Assuming the new partitions are `/dev/sdaX` and `/dev/sdaY`:

```
$ sudo pvcreate /dev/sdaX
$ sudo pvcreate /dev/sdaY
$ sudo pvdisk
```

3. 

```
$ sudo vgcreate myvg /dev/sdaX /dev/sdaY
$ sudo vgdisplay
```

4. 

```
$ sudo lvcreate -L 300M -n mylvm myvg
$ sudo lvdisplay
```

5. 

```
$ sudo mkfs.ext4 /dev/myvg/mylvm
$ sudo mkdir /mylvm
$ sudo mount /dev/myvg/mylvm /mylvm
```

If you want the mount to be persistent, edit `/etc/fstab` to include the line:



in `/etc/fstab`

```
/dev/myvg/mylvm /mylvm ext4 defaults 0 0
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

```
6. $ sudo lvdisplay

7. $ df -h
$ sudo lvresize -r -L 350M /dev/myvg/mylvm
$ df -h
```

or

```
$ sudo lvresize -r -L +50M /dev/myvg/mylvm
```

or with older methods you can do:

```
$ df -h
$ sudo lvextend -L 350M /dev/myvg/mylvm
$ sudo resize2fs /dev/myvg/mylvm
$ df -h
```

# Chapter 24

## RAID \*\*



|      |                                       |     |
|------|---------------------------------------|-----|
| 24.1 | RAID . . . . .                        | 292 |
| 24.2 | RAID Levels . . . . .                 | 293 |
| 24.3 | Software RAID Configuration . . . . . | 294 |
| 24.4 | Monitoring RAIDs . . . . .            | 295 |
| 24.5 | RAID Hot Spares . . . . .             | 296 |
| 24.6 | Labs . . . . .                        | 297 |



### Please Note

\*\* These sections may be considered in part or in whole as optional. They contain either background reference material, specialized topics, or advanced subjects. The instructor may choose to cover or not cover them depending on classroom experience and time constraints.

## 24.1 RAID

# RAID

- **RAID (Redundant Array of Independent Disks):**
  - Spreads I/O over multiple disks
  - Can improve both performance and reliability
  - Can be implemented in either hardware or software
  - Comes in a number of different **RAID levels**

### Essential features of RAID

**mirroring:** writing the same data to more than one disk.

**striping:** splitting of data to more than one disk

**parity:** extra data is stored to allow problem detection and repair

**RAID (Redundant Array of Independent Disks)** spreads I/O over multiple disks. This can really increase performance in modern disk controller interfaces, such as **SCSI**, which can perform the work in parallel efficiently.

**RAID** can be implemented either in software (it is a mature part of the **Linux** kernel) or in hardware. If your hardware **RAID** is known to be of good quality it should be more efficient than using software **RAID**. With the hardware implementation the operating system is actually not directly aware of using **RAID**. For example, three 512 GB hard drives (two for data, one for parity) configured with RAID-5 will just look like a single 1 TB disk.

However, one disadvantage of using hardware **RAID** is that if the disk controller fails, it must be replaced by a compatible controller which may not be easy to obtain. When using software **RAID**, the same disks can be attached to and work with any disk controller. Such considerations are more likely to be relevant for low and mid-range hardware.

One of the main purposes of a **RAID** is to create a filesystem which spans more than one disk. This allows us to create filesystems which are larger than any one drive. **RAID** devices are typically created by combining partitions from several disks together.

Another advantage of **RAID** devices is the ability to provide better performance, redundancy, or both. Striping provides better performance by spreading the information over multiple devices so simultaneous writes are possible. Mirroring writes the same information to multiple drives giving better redundancy.

**mdadm** is used to create and manage **RAID** devices.

Once created, the array name, `/dev/mdX` can be used just like any other device, such as `/dev/sda1`.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 24.2 RAID Levels

# RAID Levels

- **RAID 0**: striping
- **RAID 1**: mirroring
- **RAID 5**: striping with distributed parity
- **RAID 6**: striping with dual distributed parity
- **RAID 10**: mirrored and striped
- **RAID** levels can be combined

There are a number of **RAID** specifications of increasing complexity and use. The most commonly used are levels 0, 1, and 5.

- **RAID 0** uses only striping. Data is spread across multiple disks. However, in spite of the name, there is no redundancy and there is no stability or recovery capabilities. In fact, if any disk fails data will be lost. But performance can be improved significantly because of parallelization of I/O tasks.
- **RAID 1** uses only mirroring; each disk has a duplicate. This is good for recovery. At least two disks are required.
- **RAID 5** uses a rotating parity stripe; a single drive failure will cause no loss of data, only a performance drop. There must be at least 3 disks.
- **RAID 6** has striped disks with dual parity; it can handle loss of two disks, and requires at least 4 disks. Because **RAID 5** can impose significant stress on disks, which can lead to failures during recovery procedures, **RAID 6** has become more important.
- **RAID 10** is a mirrored and striped data set. At least 4 drives are needed.

As a general rule adding more disks improves performance.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 24.3 Software RAID Configuration

# Software RAID Configuration

Essential steps in configuring a software **RAID** device:

- Create partitions on each disk (type fd in **fdisk**)
- Create **RAID** device with **mdadm**
- Format **RAID** device
- Add device to **/etc/fstab**
- Mount **RAID** device
- Capture **RAID** details to ensure persistence

The command:

```
$ sudo mdadm -S
```

can be used to stop **RAID**.

For example, create two partitions of type fd on disks sdb and sdc (say **/dev/sdbX** and **/dev/sdcX**) by running **fdisk** on each:

```
$ sudo fdisk /dev/sdb
$ sudo fdisk /dev/sdc
```

Then set up the array, format it, add to the configuration and mount it:

```
$ sudo mdadm --create /dev/md0 --level=1 --raid-disks=2 /dev/sdbX /dev/sdcX
$ sudo mkfs.ext4 /dev/md0
$ sudo bash -c "mdadm --detail --scan >> /etc/mdadm.conf"
$ sudo mkdir /myraid
$ sudo mount /dev/md0 /myraid
```

Be sure to add a line in **/etc/fstab** for the mount point.

```
/dev/md0 /myraid ext4 defaults 0 2
```

You can examine **/proc/mdstat** to see the **RAID** status as in:

```
$ cat /proc/mdstat
Personalities : [raid1]
md0 : active raid1 sda8[1] sda7[0]
----- 521984 blocks [2/2]
unused devices: <none>
```

Use **mdadm -S /dev/md0** to stop the RAID device.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 24.4 Monitoring RAIDs

# Monitoring RAIDs

- Monitor RAID devices multiple ways

```
$ sudo mdadm --detail /dev/md0
$ cat /proc/mdstat
```

- **mdmonitor**

```
/etc/mdadm.conf
```

You can monitor **RAID** devices multiple ways as in:

```
$ sudo mdadm --detail /dev/md0
$ cat /proc/mdstat
```

One can also use **mdmonitor** which requires configuring `/etc/mdadm.conf`.

The command:

```
$ sudo mdadm --detail /dev/mdX
```

will show the current status of the **RAID** device `/dev/mdX`. Another way to do this is to examine the `/proc` filesystem:

```
$ cat /proc/mdstat
```

will show the status of all **RAID** devices on the system.

You can also use the **mdmonitor** service by editing `/etc/mdadm.conf` and adding a line like:

```
MAILADDR eddie@haskell.com
```

so that it notifies you with email sent to `eddie@haskell.com` when a problem occurs with a **RAID** device, such as when any of the arrays fail to start or fall into a degraded state. You turn it on with:

```
$ sudo systemctl start mdmonitor
```

and make sure it starts at boot with:

```
$ sudo systemctl enable mdmonitor
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 24.5 RAID Hot Spares

# RAID Hot Spares

- Unused RAID member set aside as a hot spare
  - Can be created along with the array
  - Can be added after-the-fact
- Testable

```
$ mdadm --fail
$ mdadm --remove
$ mdadm --add
```

One of the important things that **RAID** provides is redundancy. To help ensure any reduction in that redundancy is fixed as quick as possible, a **hot spare** can be used.

To create a hot spare when creating the **RAID** array:

```
$ sudo mdadm --create /dev/md0 -l 5 -n3 -x 1 /dev/sda8 /dev/sda9 /dev/sda10 /dev/sdb2
```

The **-x 1** switch tells **mdadm** to use one spare device. Note a hot spare can also be added at a later time.

The command:

```
$ sudo mdadm --fail /dev/md0 /dev/sdb2
```

will test the redundancy and hot spare of your array.

To restore the tested drive, or a new drive in a legitimate failure situation, first remove the “faulty” member, then add the “new” member as in:

```
$ sudo mdadm --remove /dev/md0 /dev/sdb2
$ sudo mdadm --add /dev/md0 /dev/sde2
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 24.6 Labs

### Exercise 24.1: Creating a RAID Device

Normally when creating a **RAID** device we would use partitions on separate disks. However, for this exercise we probably don't have such hardware available. Thus we will need to have two partitions on the same disk.

The process will be the same whether the partitions are on one drive or several (Although there is obviously little reason to actually create a **RAID** on a single device).

1. Create two 200 MB partitions of type raid (fd) either on your hard disk using **fdisk**, or using **LVM**.
2. Create a **RAID 1** device named `/dev/md0` using the two partitions.
3. Format the **RAID** device as an **ext4** filesystem. Then mount it at `/myraid` and make the mount persistent.
4. Place the information about `/dev/md0` in `/etc/mdadm.conf`, using **mdadm**. (Depending on your distribution, this file may not previously exist.)
5. Examine `/proc/mdstat` to see the status of your **RAID** device.

### Solution 24.1

1. If you need to create new partitions do:

```
$ sudo fdisk /dev/sda
```

and create the partitions as we have done before. For purposes of being definite, we will call them `/dev/sdAX` and `/dev/sdAY`. You will need to run **partprobe** or **kpartx** or reboot after you are done to make sure the system is properly aware of the new partitions.

2. 

```
$ sudo mdadm -C /dev/md0 --level=1 --raid-disks=2 /dev/sdAX /dev/sdAY
```
3. 

```
$ sudo mkfs.ext4 /dev/md0
$ sudo mkdir /myraid
$ sudo mount /dev/md0 /myraid
```

and add to `/etc/fstab`



in `/etc/fstab`

```
/dev/md0 /myraid ext4 defaults 0 0
```

4. 

```
$ sudo bash -c "mdadm --detail --scan >> /etc/mdadm.conf"
```

5. 

```
$ cat /proc/mdstat
```

```
1 Personalities : [raid1]
2 md0 : active raid1 dm-14[1] dm-13[0]
3 204736 blocks [2/2] [UU]
4
5 unused devices: <none>
```



Please Note

You should probably verify that with a reboot, the **RAID** volume is mounted automatically. When you are done, you probably will want to clean up by removing the line from `/etc/fstab`, and then getting rid of the partitions.

For the exclusive use of LFS301 corp class taught 06 to 09 December

---

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 25

# Kernel Services and Configuration



|      |                                  |     |
|------|----------------------------------|-----|
| 25.1 | Kernel Overview . . . . .        | 300 |
| 25.2 | Kernel Configuration . . . . .   | 301 |
| 25.3 | Kernel Boot Parameters . . . . . | 302 |
| 25.4 | sysctl . . . . .                 | 303 |
| 25.5 | Labs . . . . .                   | 304 |

## 25.1 Kernel Overview

# Kernel Overview

- The kernel is the core of the operating system
- Kernel responsibilities include:
  - System initialization and boot up
  - Process scheduling
  - Memory management
  - Controlling access to hardware
  - I/O (Input/Output) between applications and storage devices
  - Implementation of local and network filesystems
  - Security control, both locally (such as filesystem permissions) and over the network
  - Networking control
- The operating system contains many other components in addition to the kernel

Narrowly defined, **Linux** is only the **kernel** of the **operating system**, which includes many other components, such as libraries and applications that interact with the kernel.

The kernel is the essential central component that connects the hardware to the software and manages system resources, such as memory and CPU time allocation among competing applications and services. It handles all connected devices using **device drivers**, and makes the devices available for operating system use.

A system running **only** a kernel has rather limited functionality; it will be found only in dedicated and focused **embedded devices**.

## 25.2 Kernel Configuration

### Kernel Command Line

- Parameters passed to the system at boot on the **kernel command line**
- Placed on the `linux` or `linux16` line in the **GRUB** configuration file: `grub.cfg` somewhere under `/boot`
- Can be modified interactively at boot
- Exact appearance will depend on **Linux** distribution and version:

```
linux /boot/vmlinuz-4.15.0-58-generic \
 root=UUID=5cec328b-bcd6-46d2-bcfa-8430257cd7a5 \
 ro find_preseed=/preseed.cfg auto noprompt \
 priority=critical locale=en_US quiet crashkernel=512M-:192M
```

or

```
$ cat /proc/cmdline

BOOT_IMAGE=/boot/vmlinuz-4.15.0-58-generic \
root=UUID=5cec328b-bcd6-46d2-bcfa-8430257cd7a5\
ro find_preseed=/preseed.cfg auto noprompt \
priority=critical locale=en_US quiet crashkernel=512M-:192M
```

Sample kernel command lines will depend on distributions and might look like:

```
linux /boot/vmlinuz-4.19.0 root=UUID=7ef4e747-afae-48e3-90b4-9be8be8d0258 ro quiet crashkernel=384M-:128M
```

```
linuxefi /boot/vmlinuz-5.2.9 root=UUID=77461ee7-c34a-4c5f-b0bc-29f4feecc743 ro crashkernel=auto rhgb quiet
```

and would be found in `grub.cfg` in a subdirectory under `boot`, such as `/boot/grub` or in a place like `/boot/efi/EFI/centos/grub.cfg`.

Everything after the `vmlinuz` file specified is an option. Any options not understood by the kernel will be passed to `init` (`pid = 1`) the first user process to be run on the system. To see what command line a system was booted with type:

```
$ cat /proc/cmdline
```

```
BOOT_IMAGE=(hd0,msdos1)/boot/vmlinuz-5.11.0 root=UUID=7a8244d5-f289-4067-8ad6-9090080b7e35 ro
↪ resume=UUID=d602c4e1-ef8a-4945-8e3b-e98fcc8bfba2 rhgb quiet
```



#### BLSFG Changes things

**Fedora** and **RHEL/CentOS 8** now use **BLSFG** (Boot Loader Specification Configuration) which alters where the kernel command line is set. You should now look in `/boot/grub2/grubenv` for that information.

We will discuss this later in more detail when we take up `grub` in a full chapter.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 25.3 Kernel Boot Parameters

# Kernel Boot Parameters

- Appear on the kernel line in `grub.cfg`
- Can also be supplied interactively at boot
- Parameters can be specified either as:
  - a simple value as an argument
  - in the form `param=value` where `value` can be a string, integer, array of integers etc.

```
vmlinuz root=/dev/sda6 noapic crashkernel=256M
```

- No intentionally secret, hidden, parameters
- Enumerated in `kernel-parameters.txt` in the kernel source

Kernel options are placed at the end of the kernel line and are separated by spaces. An example of kernel boot parameters (all on one line):

```
linux16 /boot/vmlinuz-3.19.1.0 root=UUID=178d0092-4154-4688-af24-cda272265e08 ro
vconsole.keymap=us crashkernel=auto vconsole.font=latacyrheb-sun16 rhgb quiet LANG=en_US.UTF-8
```

- `root`: root filesystem
- `ro`: mounts root device read-only on boot
- `vconsole.keymap`: which keyboard to use on the console
- `crashkernel`: how much memory to set aside for **kernel crashdumps**
- `vconsole.font`: which font to use on the console
- `rhgb`: for graphical boot
- `quiet`: disables most log messages
- `LANG`: is the system language

See `man bootparam` and `kernel-parameters.txt` in the kernel documentation. Kernel documentation can be found in a package with a name like `kernel-doc` or `linux-doc`, or online at <http://kernel.org/doc/Documentation/>.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 25.4 sysctl

# sysctl

- Settings in `/proc/sys` can be manipulated with **sysctl**
- Variable names correspond to paths in `/proc/sys`
- Can view, set, or change kernel parameters

```
File Edit View Search Terminal Help
[student@CentOS7 ~]$ sysctl -a
...
kernel.pid_max = 131072
...
kernel.tainted = 0
kernel.threads-max = 29215
...
net.ipv4.ip_default_ttl = 64
...
net.ipv4.ip_forward = 1
...
vm.nr_hugepages = 0
...
vm.swappiness = 30
...
[student@CentOS7 ~]$
c7:/tmp>]
```

Figure 25.1: **sysctl**

The **sysctl** interface can be used to read and tune kernel parameters at run time. Current values can be displayed by doing:

```
$ sysctl -a
```

The following two statements are equivalent:

```
$ sudo sh -c 'echo 1 > /proc/sys/net/ipv4/ip_forward'
$ sudo sysctl net.ipv4.ip_forward=1
```

(Do not leave spaces around the = sign.) Note in the first form, we can not just use a simple **sudo** with **echo**; the command must be done in the complicated way shown, or executed as root.

If settings are placed in `/etc/sysctl.conf` settings can be fixed at boot time. Note that typing:

```
$ sudo sysctl -p
```

effectuates immediate digestion of the file, setting all parameters as found; this is also part of the boot process.



`/usr/lib/sysctl.d` and `/usr/lib/sysctl.d`

With the advent of **systemd**, things are a little more complicated. Vendors put their settings in files the `/usr/lib/sysctl.d/` directory. These can be added to or supplanted by files placed in `/etc/sysctl.d`. However, the original file (`/etc/sysctl.conf`) is still supported as is self-documented in that file.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 25.5 Labs



### Video Demonstration Resources

[using\\_sysctl\\_demo.mp4](#)

#### Exercise 25.1: System Tunables with sysctl

1. Check if you can **ping** your own system.
2. Check the current value of `net.ipv4.icmp_echo_ignore_all`, which is used to turn on and off whether your system will respond to **ping**. A value of 0 allows your system to respond to pings.
3. Set the value to 1 using the **sysctl** command line utility and then check if pings are responded to.
4. Set the value back to 0 and show the original behavior is restored.
5. Now change the value by modifying `/etc/sysctl.conf` and force the system to activate this setting file without a reboot.
6. Check that this worked properly.

You will probably want to reset your system to have its original behavior when you are done.

#### Solution 25.1

You can use either `localhost`, `127.0.0.1` (loopback address) or your actual IP address for target of **ping** below.

1. `$ ping localhost`
2. `$ sysctl net.ipv4.icmp_echo_ignore_all`
3. `$ sudo sysctl net.ipv4.icmp_echo_ignore_all=1`  
`$ ping localhost`
4. `$ sudo sysctl net.ipv4.icmp_echo_ignore_all=0`  
`$ ping localhost`
5. Add the following line to `/etc/sysctl.conf`:



in `/etc/sysctl.conf`

`net.ipv4.icmp_echo_ignore_all=1`

and then do:

`$ sysctl -p`

6. `$ sysctl net.ipv4.icmp_echo_ignore_all`  
`$ ping localhost`

Since the changes to `/etc/sysctl.conf` are persistent, you probably want to restore things to its previous state.

#### Exercise 25.2: Changing the Maximum Process ID

The normal behavior of a **Linux** system is that process IDs start out at `PID=1` for the **init** process, the first user process on the system, and then go up sequentially as new processes are constantly created (and die as well.)

For the exclusive use of LFS301 corp class taught 06 to 09 December

However, when the PID reaches the value shown in `/proc/sys/kernel/pid_max`, which is conventionally 32768 (32K), they will wrap around to lower numbers. If nothing else, this means you can't have more than 32K processes on the system since there are only that many slots for PIDs.

1. Obtain the current maximum PID value.
2. Find out what current PIDs are being issued
3. Reset `pid_max` to a lower value than the ones currently being issued.
4. Start a new process and see what it gets as a PID.

## ✓ Solution 25.2



### Very Important

In the below we are going to use two methods, one involving `sysctl`, the other directly echoing values to `/proc/sys/kernel/pid_max`. Note that the `echo` method requires you to be root; `sudo` won't work. We'll leave it to you to figure out why, if you don't already know!

1. 

```
$ sysctl kernel.pid_max
$ cat /proc/sys/kernel/pid_max
```
2. Type:  

```
$ cat &
1 [1] 29222
```

```
$ kill -9 29222
```
3. 

```
$ sudo sysctl kernel.pid_max=24000
$ echo 24000 > /proc/sys/kernel/pid_max # This must be done as root
$ cat /proc/sys/kernel/pid_max
```
4. 

```
$ cat &
1 [2] 311
```

```
$ kill -9 311
```



### Please Note

Note that when starting over, the kernel begins at PID=300, not a lower value. You might notice that assigning PIDs to new processes is actually not trivial; since the system may have already turned over, the kernel always has to check when generating new PIDs that the PID is not already in use. The **Linux** kernel has a very efficient way of doing this that does not depend on the number of processes on the system.

For the exclusive use of LFS301 corp class taught 06 to 09 December

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 26

# Kernel Modules



|      |                                |     |
|------|--------------------------------|-----|
| 26.1 | Kernel Modules . . . . .       | 308 |
| 26.2 | Module Utilities . . . . .     | 310 |
| 26.3 | modinfo . . . . .              | 312 |
| 26.4 | Module Configuration . . . . . | 313 |
| 26.5 | Labs . . . . .                 | 314 |

## 26.1 Kernel Modules

# Kernel Modules

- Extensions to the kernel
- May be loaded or unloaded dynamically, as needed
- Often embody device drivers, network protocols, etc
- Located in `/lib/modules/$(uname -r)`
- Compiled for specific kernel versions
- Kernel remains a **monolithic** one, not a **microkernel**

The **Linux** kernel makes extensive use of **modules**, which contain important software that can be loaded and unloaded as needed after the system starts.

Many modules incorporate **device drivers** to control hardware either inside the system or attached peripherally. Other modules can control network protocols, support different filesystem types and many other purposes.

Parameters can be specified when loading modules to control their behavior. The end result is great flexibility and agility in responding to changing conditions and needs.

## Listing Modules in Use with lsmod

```
c8:/tmp>lsmod
Module Size Used by
vmnet 57344 13
vmmon 126976 0
rfcomm 90112 4
nft_fib_inet 16384 1
bnep 28672 2
vmw_vsock_vmci_transport 32768 0
vmw_vmci 81920 1 vmw_vsock_vmci_transport
kvm_intel 270336 0
kvm 958464 1 kvm_intel
irqbypass 16384 1 kvm
ext4 905216 5
mbcache 16384 1 ext4
jbd2 147456 1 ext4
e1000e 290816 0
c8:/tmp>
```

Figure 26.1: Using lsmod

Most kernel features can be configured as modules, even if they are almost always likely to be used. This flexibility also aids in development of new features as system reboots are almost never needed to test during development and debugging.

While other operating systems have used module-like methods, **Linux** uses it far more than any other operating system.

## 26.2 Module Utilities

# Module Utilities

- **lsmod**  
list loaded modules
- **insmod**  
directly load modules
- **rmmod**  
directly remove modules
- **modprobe**  
load or unload modules using a pre-built database with dependency and location information
- **depmod**  
rebuild the module dependency database
- **modinfo**  
display information about a module

Use **modprobe** to load a module:

```
$ modprobe e1000e
```

Use **modprobe -r** to unload or remove a module:

```
$ modprobe -r e1000e
```

**modprobe** requires a module dependency database be updated. Use **depmod** to generate or update the file `/lib/modules/$(uname -r)/modules.dep`.

Use **insmod** to directly load a module (requires fully qualified module name):

```
$ insmod /lib/modules/$(uname -r)/kernel/drivers/net/ethernet/intel/e1000e.ko
```

Use **rmmod** to directly remove a module:

```
$ rmmod e1000e
```

Use **lsmod** to list the loaded modules:

```
$ lsmod
```

Use **modinfo** to display information about a module (including parameters):

```
$ modinfo e1000e
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Some Considerations with Modules

- It is impossible to unload a module:
  - In use by another module
  - In use by a running process
- It is impossible to load a module:
  - Compiled for a different kernel, or with different options
  - That conflicts with a currently running module
- Modules are usually loaded with **modprobe**, not **insmod**
- Modules loaded with non-acceptable open source licenses mark the kernel as **tainted**

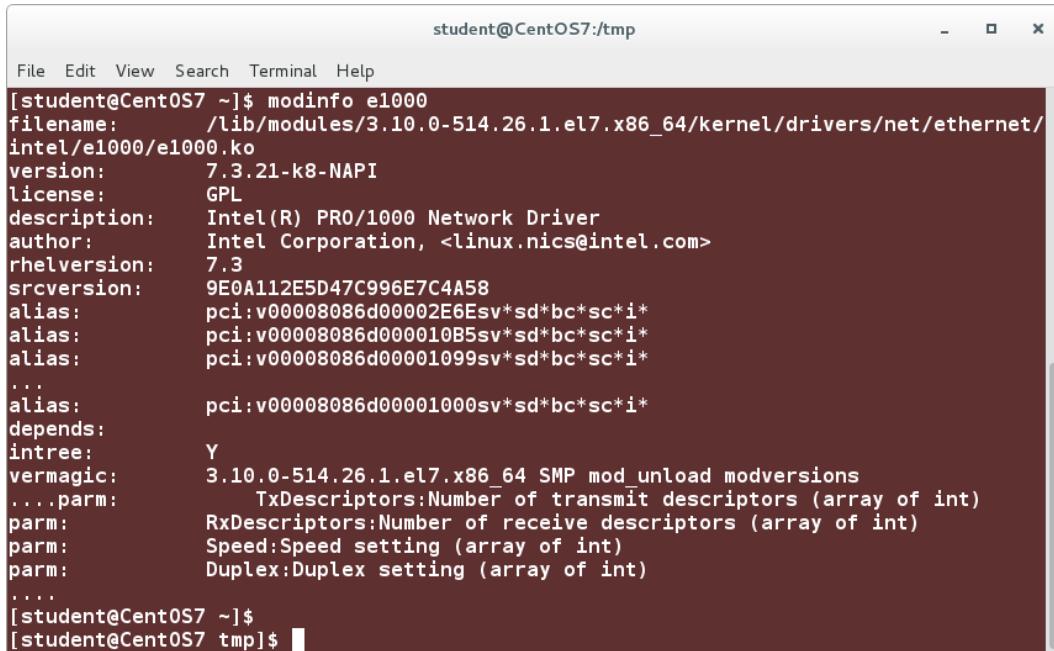
There are some important things to keep in mind when loading and unloading modules:

- It is impossible to unload a module being used by one or more other **modules**, which one can ascertain from the **lsmod** listing.
- It is impossible to unload a module that is being used by one or more **processes**, which can also be seen from the **lsmod** listing. However, there are modules which do not keep track of this reference count, such as network device driver modules, as it would make it too difficult to temporarily replace a module without shutting down and restarting much of the whole network stack.
- When a module is loaded with **modprobe**, the system will automatically load any other modules that need to be loaded first.
- When a module is unloaded with **modprobe -r**, the system will automatically unload any other modules being used by the module, if they are not being simultaneously used by any other loaded modules.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 26.3 modinfo

### modinfo Example



```
student@CentOS7:tmp
File Edit View Search Terminal Help
[student@CentOS7 ~]$ modinfo e1000
filename: /lib/modules/3.10.0-514.26.1.el7.x86_64/kernel/drivers/net/ethernet/intel/e1000/e1000.ko
version: 7.3.21-k8-NAPI
license: GPL
description: Intel(R) PRO/1000 Network Driver
author: Intel Corporation, <linux.nics@intel.com>
rhelversion: 7.3
srcversion: 9E0A112E5D47C996E7C4A58
alias: pci:v00008086d00002E6Esv*sd*bc*sc*i*
alias: pci:v00008086d000010B5sv*sd*bc*sc*i*
alias: pci:v00008086d00001099sv*sd*bc*sc*i*
...
alias: pci:v00008086d00001000sv*sd*bc*sc*i*
depends:
intree: Y
vermagic: 3.10.0-514.26.1.el7.x86_64 SMP mod_unload modversions
....parm: TxDescriptors:Number of transmit descriptors (array of int)
parm: RxDescriptors:Number of receive descriptors (array of int)
parm: Speed:Speed setting (array of int)
parm: Duplex:Duplex setting (array of int)
...
[student@CentOS7 ~]$
[student@CentOS7 tmp]$
```

Figure 26.2: **modinfo**

**modinfo** can be used to find out information about kernel modules (whether or not they are currently loaded) as shown in the above screenshot, which displays information about version, file name, which hardware devices the device driver module can handle, and what parameters can be supplied on loading.

Much information about modules can also be seen in the `/sys` pseudo-filesystem directory tree; in the above example one would look under `/sys/module/e1000` and some if not all parameters can be read and/or written under `/sys/module/e1000/parameters`. We will show how to set them next.

Many modules can be loaded while specifying parameter values, such as in:

```
$ sudo /sbin/insmod <path to>/e1000e.ko debug=2 copybreak=256
```

or, for a module already in the proper system location, it is easier with:

```
$ sudo /sbin/modprobe e1000e debug=2 copybreak=256
```

## 26.4 Module Configuration

### /etc/modprobe.d/

- All files in the `/etc/modprobe.d` subdirectory tree which end with the `.conf` extension are scanned when modules are loaded and unloaded using **modprobe**
- Define aliases for module names
- Can **blacklist** particular modules
- Define parameters to be passed to module at load time
- Specify commands to be run when modules are loaded or unloaded

Files in the `/etc/modprobe.d` directory control some parameters that come into play when loading with **modprobe**. These parameters include module name **aliases** and automatically supplied options. One can also **blacklist** specific modules to avoid them being loaded.

Settings apply to modules as they are loaded or unloaded, and configurations can be changed as needs change.

The format of files in `/etc/modprobe.d` is simple: one command per line, with blank lines and lines starting with # ignored (useful for adding comments). A backslash at the end of a line causes it to continue on the next line, which makes the file a bit neater.

## 26.5 Labs



### Video Demonstration Resources

[using\\_kernel\\_modules.mp4](#)

#### Exercise 26.1: Kernel Modules

1. List all currently loaded kernel modules on your system.

2. Load a currently unloaded module on your system.

If you are running a distribution kernel, this is easy to find; you can simply look in the `/lib/modules/(kernel-version)/kernel/drivers/net` directory and grab one. (Distribution kernels come with drivers for every device, filesystem, network protocol etc. that a system might need.) However, if you are running a custom kernel you may not have many unloaded modules compiled.

A choice that will usually work is to pick either `e1000.ko` or `e1000e.ko`, as while these gigabit Ethernet drivers are quite common, it is very unlikely both would be loaded at once.

3. Re-list all loaded kernel modules and see if your module was indeed loaded.

4. Remove the loaded module from your system.

5. Re-list again and see if your module was properly removed.

#### Solution 26.1

1. `$ lsmod`

2. In the following, substitute whatever module name you used for `e1000e`. Either of these methods work but, of course, the second is easier.

```
$ sudo insmod /lib/modules/$(uname -r)/kernel/drivers/net/ethernet/intel/e1000e.ko
$ sudo /sbin/modprobe e1000e
```

3. `$ lsmod | grep e1000e`

4. Once again, either method works.

```
$ sudo rmmod e1000e
$ sudo modprobe -r e1000e
```

5. `$ lsmod | grep e1000e`

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 27

# Devices and udev



|      |                                      |     |
|------|--------------------------------------|-----|
| 27.1 | udev and Device Management . . . . . | 316 |
| 27.2 | Device Nodes . . . . .               | 317 |
| 27.3 | Rules . . . . .                      | 320 |
| 27.4 | Labs . . . . .                       | 323 |

## 27.1 udev and Device Management

# udev

- **udev** stands for **User Device** management
- **udev** dynamically discovers built in hardware as well as peripheral devices:
  - during system boot
  - when **hotplugged** at any time
- **udev** handles loading and unloading device drivers with proper configurations as need
- **Device Nodes** are created automatically and then used by applications and operating system subsystems
- System administrators can control how **udev** operates and craft special **udev rules**.

**udev** actions include:

- Device naming
- Device file and symlink creating
- Setting file attributes
- Taking needed actions

When devices are added or removed from the system, **udev** receives a message from the kernel. It then parses the rules files in the `/etc/udev/rules.d` directory to see if any rules are there for the device added or removed.

These rules are totally customizable and can specify device file names, device file creation, specify symlinks to be created, specify file attributes to be set for the device file (including user and group ownership), and even specify actions to be taken (programs to be executed).

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 27.2 Device Nodes

# Device Nodes Connect Hardware with Software

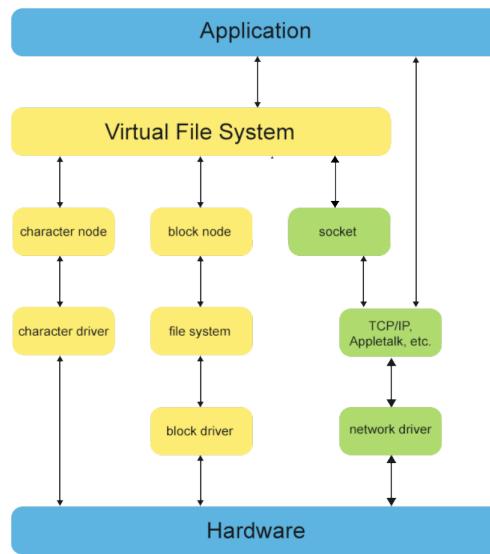


Figure 27.1: Device Nodes

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Device Nodes

- **Device Nodes** are located in the `/dev` directory
- Character and Block devices have device nodes; Network devices do not
- Programs communicate with devices through these nodes using normal I/O methods
- One device driver may use multiple device nodes
- Device nodes can be created with:
 

```
$ sudo mknod [-m mode] /dev/name <type> <major> <minor>
```

 e.g.,
 

```
$ sudo mknod -m 666 /dev/mycdrv c 254 1
```

```
$ ls -l /dev
1 total 0
2 crw----- 1 coop audio 14, 4 Jul 9 01:54 audio
3 crw----- 1 root root 10, 62 Jul 9 01:54 autofs
4 lrwxrwxrwx 1 root root 4 Jul 9 01:54 cdrom -> scd0
5 lrwxrwxrwx 1 root root 4 Jul 9 01:54 cdrw -> scd0
6 crw----- 1 coop root 5, 1 Jul 9 06:54 console
7
8 lrwxrwxrwx 1 root root 4 Jul 9 01:54 dvd -> scd0
9 lrwxrwxrwx 1 root root 4 Jul 9 01:54 dvdwriter -> scd0
10
11 brw-r---- 1 root disk 8, 0 Jul 9 01:53 sda
12 brw-r---- 1 root disk 8, 1 Jul 9 01:53 sda1
13 brw-r---- 1 root disk 8, 2 Jul 9 06:54 sda2
14
15 brw-r---- 1 root disk 8, 16 Jul 9 01:53 sdb
16 brw-r---- 1 root disk 8, 17 Jul 9 01:53 sdb1
17 brw-r---- 1 root disk 8, 18 Jul 9 01:53 sdb2
18
19 crw-rw-rw- 1 root tty 5, 0 Jul 9 01:54 tty
20 crw-rw---- 1 root root 4, 0 Jul 9 14:54 tty0
21 crw----- 1 root root 4, 1 Jul 9 06:54 tty1
22 cr--r--r-- 1 root root 1, 9 Jul 9 01:53 urandom
23
24 crw-rw-rw- 1 root root 1, 5 Jul 9 01:54 zero
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

# udev Components

- **libudev** library

Allows access to information about the devices

- **udevd** or **systemd-udevd** daemon

Manages the `/dev` directory

- **udevadm**

Utility for control and diagnostics

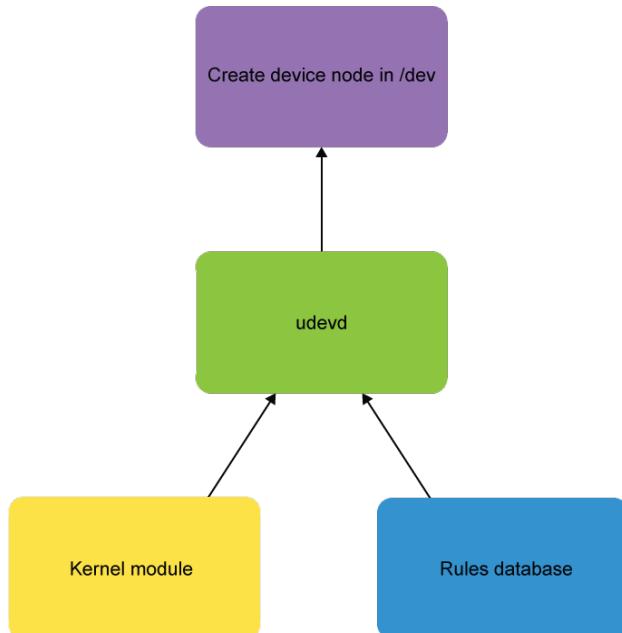


Figure 27.2: **udev Rules**

**udev** runs as a **daemon** (either **udevd** or **systemd-udevd**) and monitors a **netlink** socket. When new devices are initialized or removed, the **uevent** kernel facility sends a message through the socket, which **udev** receives and takes appropriate action to create or remove device nodes of the right names and properties according to the rules.

The three components of **udev** are:

1. The **libudev** library which allows access to information about the devices.
2. The **udevd** daemon that manages the `/dev` directory
3. The **udevadm** utility for control and diagnostics.

The cleanest way to use **udev** is to have a pure system; the `/dev` directory is empty upon the initial kernel boot, and then is populated with device nodes as they are needed. When used this way, one must boot using an **initramfs** image, which may contain a set of preliminary device nodes as well as the the **udev** infrastructure.

## 27.3 Rules

# udev Rule Files

- Filename locations and format:

`/etc/udev/rules.d/<rulename>.rules`  
`/usr/lib/udev/rules.d/<rulename>.rules`

- Examples:

`30-usb.rules`  
`90-mycustom.rules`  
`70-mouse.rules`  
`60-persistent-storage.rules`

There are two separate parts defined on a single line:

- The first part consists of one or more match pairs denoted by `==`. These try to match a device's attributes and/or characteristics to some value.
- The second part consists of one or more assignment key-value pairs that assign a value to a name, such as a file name, group assignment, even file permissions, etc.

If no matching rule is found, it uses the default device node name and other attributes.

```
$ cat /etc/udev/rules.d/99-fitbit.rules
1 SUBSYSTEM=="usb", ATTR{idVendor}=="2687", ATTR{idProduct}=="fb01", SYMLINK+="fitbit", MODE="0666"
```

```
$ cat /etc/udev/rules.d/60-vboxdrv.rules
1 KERNEL=="vboxdrv", NAME="vboxdrv", OWNER="root", GROUP="vboxusers", MODE="0660"
2 KERNEL=="vboxdrvu", NAME="vboxdrvu", OWNER="root", GROUP="root", MODE="0666"
3 KERNEL=="vboxnetctl", NAME="vboxnetctl", OWNER="root", GROUP="vboxusers", MODE="0660"
4 SUBSYSTEM=="usb_device", ACTION=="add", RUN+="/usr/lib/virtualbox/VBoxCreateUSBNode.sh \
 $major $minor $attr{bDeviceClass}"
5 SUBSYSTEM=="usb", ACTION=="add", ENV{DEVTYPE}=="usb_device", RUN+="/usr/lib/virtualbox/VBoxCreateUSBNode.sh \
 $major $minor $attr{bDeviceClass}"
6 SUBSYSTEM=="usb_device", ACTION=="remove", RUN+="/usr/lib/virtualbox/VBoxCreateUSBNode.sh --remove \
 $major $minor"
7 SUBSYSTEM=="usb", ACTION=="remove", ENV{DEVTYPE}=="usb_device", RUN+="/usr/lib/virtualbox/VBoxCreateUSBNode.sh \
 --remove $major $minor"
8 SUBSYSTEM=="usb", ACTION=="remove", ENV{DEVTYPE}=="usb_device", RUN+="/usr/lib/virtualbox/VBoxCreateUSBNode.sh \
 --remove $major $minor"
9 SUBSYSTEM=="usb", ACTION=="remove", ENV{DEVTYPE}=="usb_device", RUN+="/usr/lib/virtualbox/VBoxCreateUSBNode.sh \
 --remove $major $minor"
10 SUBSYSTEM=="usb", ACTION=="remove", ENV{DEVTYPE}=="usb_device", RUN+="/usr/lib/virtualbox/VBoxCreateUSBNode.sh \
 --remove $major $minor"
11
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Creating udev Rules

- Format for rules:

```
<match><op>value [, ...] <assignment><op>value [, ...]
```

- Samples

```
KERNEL=="sdb", NAME="my-spare-disk"
KERNEL=="sdb", DRIVER=="usb-disk", SYMLINK+="sparedisk"
KERNEL=="sdb", RUN+="/usr/bin/my-program"
KERNEL=="sdb", MODE="0660", GROUP="mygroup"
```

Rules files can be in three places, and if they have same name the priority order is:

1. `/etc/udev/rules.d`
2. `/run/udev/rules.d`
3. `/usr/lib/udev/rules.d`

The format for a **udev** rule is simple. There are two separate parts defined on a single line. The first part consists of one or more match pairs (denoted by double equal signs). These will match a device's attributes and/or characteristics to some value. The second part consists of one or more assignment key-value pairs that assign a value to a name, such as a filename, group assignment, or even file permissions.

If no matching rule is found, it uses the default device node name.

See `man udev` for explanation of the key values such as `SYMLINK` and `RUN`.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Some Examples of Rules Files

- Fitbit device:

```
$ cat /etc/udev/rules.d/99-fitbit.rules
SUBSYSTEM=="usb", ATTR{idVendor}=="2687", ATTR{idProduct}=="fb01", \
SYMLINK+="fitbit", MODE="0666"
```

- For creating crash dumps and fast kernel loading with **kdump/kexec**:

```
$ cat /usr/lib/udev/rules.d/98-kexec.rules
SUBSYSTEM=="cpu", ACTION=="add", PROGRAM="/bin/systemctl \
try-restart kdump.service"
SUBSYSTEM=="cpu", ACTION=="remove", PROGRAM="/bin/systemctl \
try-restart kdump.service"
SUBSYSTEM=="memory", ACTION=="online", PROGRAM="/bin/systemctl \
try-restart kdump.service"
SUBSYSTEM=="memory", ACTION=="offline", PROGRAM="/bin/systemctl \
try-restart kdump.service"
```

- For the **kvm** virtual machine hypervisor:

```
$ cat /usr/lib/udev/rules/.d/80-kvm.rules
KERNEL=="kvm", GROUP="kvm", MODE="0666"
```

## 27.4 Labs



### Video Demonstration Resources

[using\\_udev\\_demo.mp4](#)

#### Exercise 27.1: udev

1. Create and implement a rule on your system that will create a symlink called `myusb` when a **USB** device is plugged in.
2. Plug in a **USB** device to your system. It can be a pen drive, mouse, webcam, etc.  
Note: If you are running a virtual machine under a hypervisor, you will have to make sure the **USB** device is seen by the guest, which usually is just a mouse click which also disconnects it from the host.
3. Get a listing of the `/dev` directory and see if your symlink was created.
4. Remove the **USB** device. (If it is a drive you should always **umount** it first for safety.)
5. See if your symbolic link still exists in `/dev`.

#### Solution 27.1

1. Create a file named `/etc/udev/rules.d/75-myusb.rules` and have it include just one line of content:

```
$ cat /etc/udev/rules.d/75-myusb.rules
1 SUBSYSTEM=="usb", SYMLINK+="myusb"
```

Do not use the deprecated key value `BUS` in place of `SUBSYSTEM`, as recent versions of **udev** have removed it.

Note the name of this file really does not matter. If there was an `ACTION` component to the rule the system would execute it; look at other rules for examples.

2. Plug in a device.
3. `$ ls -lF /dev | grep myusb`
4. If the device has been mounted:

```
$ umount /media/whatever
```

where `/media/whatever` is the mount point. Safely remove the device.

5. `$ ls -lF /dev | grep myusb`

---

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 28

# Virtualization Overview



|      |                                          |     |
|------|------------------------------------------|-----|
| 28.1 | Introduction to Virtualization . . . . . | 326 |
| 28.2 | Hosts and Guests . . . . .               | 328 |
| 28.3 | Emulation . . . . .                      | 329 |
| 28.4 | Hypervisors . . . . .                    | 331 |
| 28.5 | libvirt . . . . .                        | 335 |
| 28.6 | QEMU . . . . .                           | 337 |
| 28.7 | KVM . . . . .                            | 340 |
| 28.8 | Labs . . . . .                           | 342 |

## 28.1 Introduction to Virtualization

# What is Virtualization?

- An abstraction layer:
  - A **physical** resource is replaced by a **virtual** one
- Types include:
  - Operating System (e.g., **Virtual Machine, VM**)
  - Network (e.g., **Software Defined Networking, SDN**)
  - Storage (e.g., **Network Attached Storage, NAS**)
  - Application (e.g., **containers**)
- The abstraction layer may replace operating system specific **software**, rather than **hardware**

In this section of the course we will be concerned with the creation, deployment and maintenance of **Virtual Machines (VMs)**. These are a virtualized instance of an entire operating system, and may fulfill the role of a **server** or a **desktop/workstation**. The outside world sees the VM as if it were an actual physical machine, present somewhere on the network. Applications running in the VM are generally unaware of their non-physical environment.

Other kinds of virtualization include:

- **Network:** The details of the actual physical network such as the type of hardware, routers etc. are abstracted and need not be known by software running on it and configuring it.
- **Storage:**

Multiple network storage devices are configured to look like one big storage unit such as a disk.
- **Application:**

An application is isolated into a stand alone format such as a **container**, which we will discuss later.

There are differences between physical and virtual machines which are not always easy to ignore. For example, performance tuning at a low level needs to be done (separately) on both the VM and the physical machine residing underneath it.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Virtualization History

- Implemented originally on mainframes decades ago:
  - Enables better hardware utilization
  - Operating Systems often progress more quickly than hardware
  - Microcode driven
  - Not particularly user friendly
- Virtualization technology migrated to PCs and workstations:
  - Initially done using **Emulation**
  - CPUs enhanced to support Virtualization led to boost in performance, easier configuration and more flexibility in VM installation and migration

**Virtualization** has a long proud history. From early **mainframes** to **mini-computers**, virtualization has been used for expanding limits, debugging and administration enhancements.

Today, virtualization can be found everywhere in many forms.

Each of the different forms and implementations available provide particular advantages.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 28.2 Hosts and Guests

# Hosts and Guests

- **Host:**

Underlying physical operating system managing one or more virtual machines

- **Guest:**

The **VM** which is instance of a complete operating system, running one or more applications. Sometimes called a **client**

- Guest does not care or know the specific host
- Guests can be **migrated** to another host
- Performance tuning needs to be done on host as well as guest

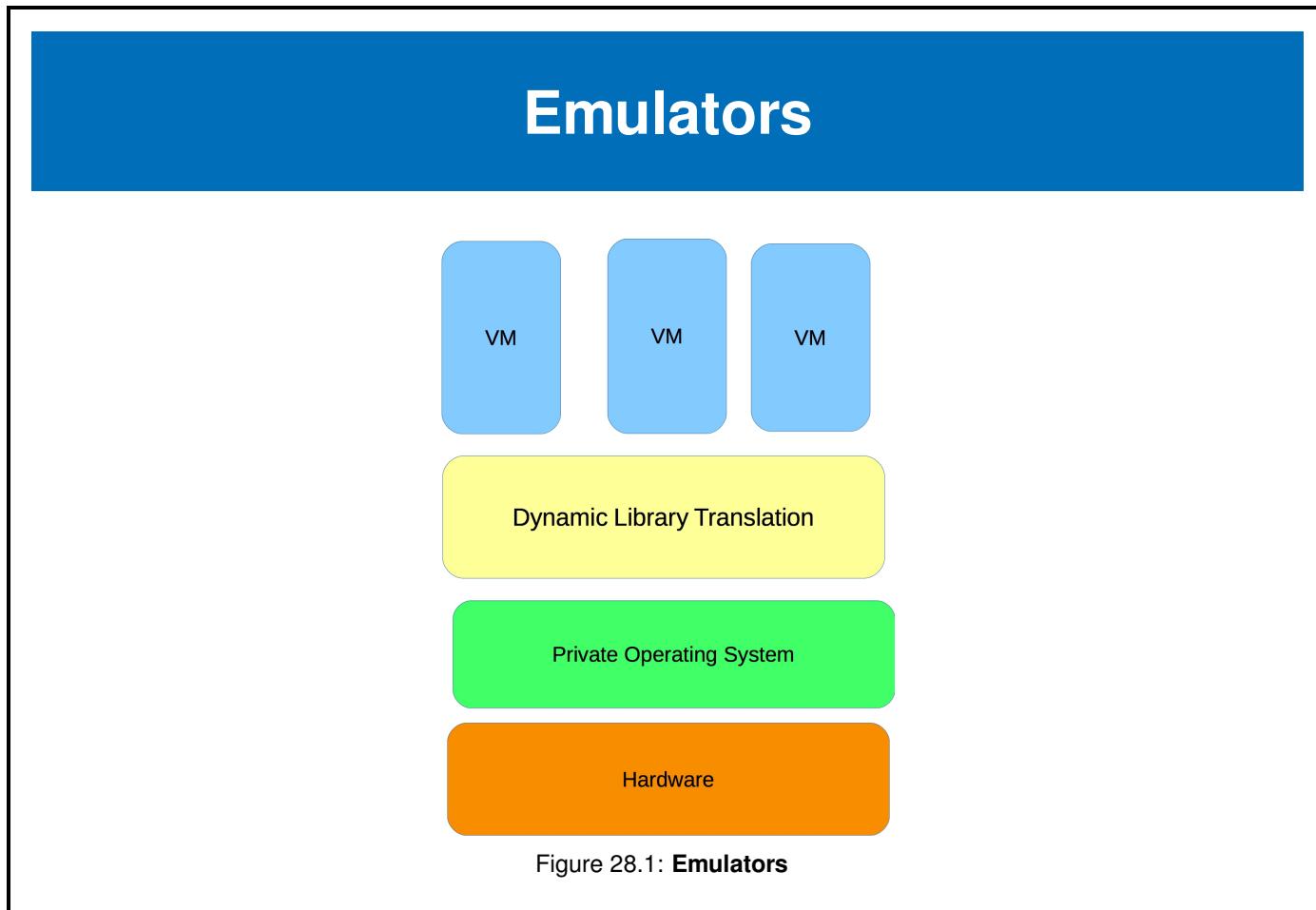
In most cases the guest should not care what host it is running on and can be **migrated** from one host to another, sometimes while actually running.

In fact, it is possible to convert physical machines into virtual ones, by copying the entire contents into a VM. Specialized software utilities can make this easier.

Low level performance tuning on areas such as CPU utilization, networking throughput, memory utilization is often best done on the host, as the guest may have only simulated qualities not directly useful.

Application tuning will be done mostly on the guest.

## 28.3 Emulation



The first implementations of virtualization on the PC architecture were through the use of **Emulators**. An application running on the current operating system would appear to another OS as a specific hardware environment. Emulators generally do not require special hardware to operate.

**Qemu** is one such emulator.

# Emulation vs Virtualization

- An **Emulator** runs completely in software
- Hardware constructs are replaced by software
- Useful for running virtual machines on different architectures: e.g., running a pretend **ARM** guest machine on an **x86** host
- Often used for developing operating system for a new CPU even before hardware is available.
- Performance is relatively slow

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 28.4 Hypervisors

# Types of Virtualization Hypervisors

- The layer that runs underneath the virtual machines and manages them
- Also called **Virtual Machine Monitor (VMM)**
- Enables multiple VMs to share hardware resources transparently
- Two basic types:
  - **Hardware Virtualization**  
Guest system is unaware it is running as a VM. Also called **Full Virtualization**
  - **Paravirtualization**  
Guest system knows it is running as a VM and has been modified to run better
- Most modern hardware has hardware virtualization abilities (must be turned on in BIOS)

The host system, besides functioning normally with respect to software that it runs, also acts as the **hypervisor** that initiates, terminates and manages guests. There are two basic methods of virtualization:

- **Hardware virtualization:** the guest system runs without being aware it is running as a virtualized guest and does not require modification to be run in this fashion.
- **Para-virtualization:** the guest system is aware it is running in a virtualized environment and has been modified specifically to work with it.

Widely-used CPUs from **Intel** and **AMD** incorporate virtualization extensions to the **x86** architecture that allow the hypervisor to run fully virtualized (i.e., unmodified) guest operating systems with only minor performance penalties.

The **Intel** extension (Intel Virtualization Technology), usually abbreviated as **VT**, **IVT**, **VT-32** or **VT-64**, is also known under the development code name **Vanderpool**. It has been available since the spring of 2005. The **AMD** extension is usually called **AMD-V** and is still sometimes referred to by the development code name **Pacifica**. For a detailed explanation and comparison of how these two extensions work see <https://lwn.net/Articles/182080/>.

You can check directly if your CPU supports hardware virtualization extensions by looking at </proc/cpuinfo>; if you have an **IVT**-capable chip you will see **vmx** in the flags field, and if you have an **AMD-V** capable chip, you will see **svm** in the same field. You may also have to ensure the virtualization capability is turned on in your **CMOS**.

While the choice of operating systems tends to be more limited for para-virtualized guests, originally they tended to run more efficiently than fully virtualized guests. Advances in virtualization techniques have narrowed or eliminated such advantages, and the wider availability of the hardware support needed for full virtualization has made para-virtualization less advantageous and less popular.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# External and in-Kernel Hypervisors

- The hypervisor can be external to the host operating system kernel:  
**VMWare**
- The hypervisor can be internal to the host operating system kernel:  
**KVM**
- In this course we will concentrate on **KVM** as it is all open source and requires no external third party hypervisor program.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Dedicated Hypervisor

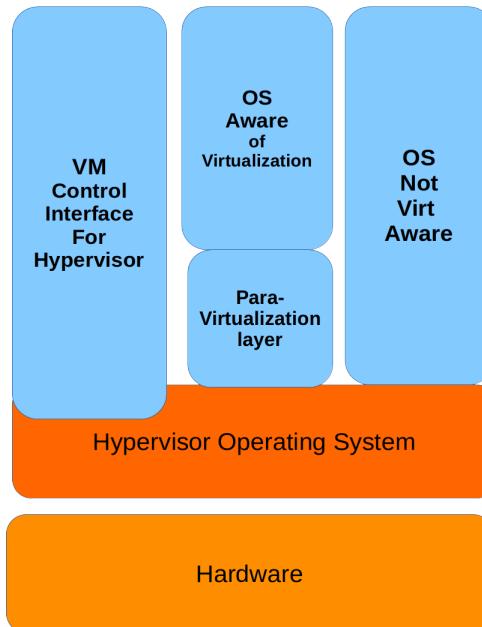


Figure 28.2: Dedicated Hypervisor

Going past Emulation, merging of the hypervisor program into a specially designed light weight kernel was the next step in Virtualization deployment.

**VMware ESX** (and related friends) is an example of a hypervisor embedded into an operating system.

# Hypervisor in the Kernel

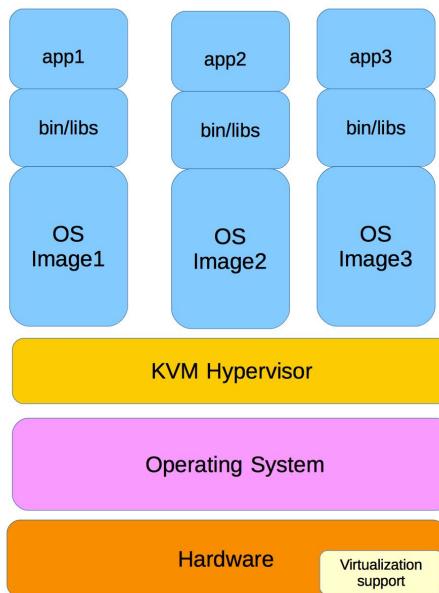


Figure 28.3: Hypervisor in the Kernel

The **KVM** project added hypervisor capabilities into the **Linux** kernel. This leveraged the facilities of the kernel and enabled the kernel to be a hypervisor.

As we have discussed, specific CPU chip functions and facilities were required and deployed for this type of virtualization.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 28.5 libvirt

# libvirt

- Toolkit to interact with virtualization technologies
- Common API interface for many hypervisors
  - KVM/QEMU
  - LXC
  - VirtualBox
  - ... and many more
- Provides management for
  - virtual machines
  - virtual networks
  - storage
- Available on all enterprise **Linux** distributions

Many application programs interface with **libvirt**, some of the most common are **virt-manager**, **virt-viewer**, **virt-install**, **virsh**. The complete list of currently supported hypervisors at <https://www.libvirt.org>:

- QEMU/KVM
- Xen
- Oracle VirtualBox
- VMware ESX
- VMware Workstation/Player
- Microsoft Hyper-V
- IBM PowerVM (phyp)
- OpenVZ
- UML (User Mode Linux)
- LXC (Linux Containers)
- Virtuozzo
- Bhyve (The BSD Hypervisor)
- Test (Used for testing)

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Programs Using libvirt

- Many utilities use **libvirt**:

```
student@ubuntu:~$ ls -lF /usr/bin/virt*
-rwxr-xr-x 1 root root 63840 Aug 9 08:50 /usr/bin/virt-admin*
-rwxr-xr-x 1 root root 57 Dec 14 2016 /usr/bin/virt-clone*
-rwxr-xr-x 1 root root 59 Dec 14 2016 /usr/bin/virt-convert*
-rwxr-xr-x 1 root root 51440 May 10 18:25 /usr/bin/virtfs-proxy-helper*
-rwxr-xr-x 1 root root 18480 Aug 9 08:50 /usr/bin/virt-host-validate*
-rwxr-xr-x 1 root root 59 Dec 14 2016 /usr/bin/virt-install*
-rwxr-xr-x 1 root root 908632 Aug 9 08:50 /usr/bin/virt-login-shell*
-rwxr-xr-x 1 root root 59 Dec 14 2016 /usr/bin/virt-manager*
-rwxr-xr-x 1 root root 11191 Aug 9 08:49 /usr/bin/virt-pki-validate*
-rwxr-xr-x 1 root root 55 Dec 14 2016 /usr/bin/virt-xml*
-rwxr-xr-x 1 root root 4700 Aug 9 08:49 /usr/bin/virt-xml-validate*
student@ubuntu:~$ \|\
```

Figure 28.4: libvirt-based Utilities

- The exact list will depend on your **Linux distribution**.
- A full list can be found at: <https://www.libvirt.org/apps.html>

In this course we will do our work through the use of the robust **GUI**, **virt-manager** rather than make much use of command line utilities, which lead to more flexibility, as well as use on non-graphical servers.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 28.6 QEMU

# What is QEMU?

- Quick **EMUlator**
- Three levels:
  1. **libvirt**: Common Library and API
  2. **QEMU**: Emulator
  3. **KVM**: Accelerator
- Hypervisor doing hardware emulation
- Host and guest can be different architectures (CPUs)
- Combine with accelerator (e.g., **KVM**) to speed up to native speeds
- Useful for development for new processing chips

**QEMU** stands for **Quick EMUlator**. It was originally written by Fabrice Bellard in 2002. (Bellard is also known for feats such as holding, at one point, the world record for calculating  $\pi$ , reaching 2.7 trillion digits.)

**QEMU** is a **hypervisor** that performs hardware **emulation**, or **virtualization**. It emulates CPUs by dynamically translating binary instructions between the host architecture and the emulated one.

The host and emulated architectures may be different or the same. There are numerous choices for both host and guest operating systems.

**QEMU** can also be used to emulate just particular applications, not entire operating systems.

By itself, **QEMU** is much slower than the host machine. But it can be used together with **KVM** (Kernel Virtual Machine) to reach speeds close to those of the native host.

Guest operating systems do not require rewriting to run under **QEMU**.

**QEMU** can save, pause, and restore a virtual machine at any time.

**QEMU** is free software licensed under the **GPL**.

**QEMU** has the capability of supporting many architectures including:

**IA-32 (i386), x86-64, MIPS, SPARC, ARM, SH4, PowerPC, CRIS, MicroBlaze, etc.**

The cross-compilation abilities of **QEMU** make it extremely useful when doing development for embedded processors.

In fact, **QEMU** has often been used to develop for processors which have either not yet been physically produced, or released to market.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Third Party Hypervisor Integration

- **QEMU** is slow but can reach almost native speed with an **accelerator** such as:
  - **KVM**
  - **Xen**
  - **Oracle Virtual Box**
- **KVM** is very tightly integrated with **QEMU** and arose from the same projects.

Used by itself, **QEMU** is relatively slow. However, it can be integrated with third party hypervisors, and then reach near native speeds. Note some of these systems are very close cousins of **QEMU**; others are more remote. To mention a few main ones:

- **KVM** offers particularly tight integration with **QEMU**. When host and target are the same architecture, full acceleration and high speed are delivered. **KVM** is native to **Linux**. We will discuss this in detail.
- **Xen**, also native to **Linux**, can run in hardware virtualization mode if the architecture offers it, as does **x86** and some **ARM** variants.
- **Oracle Virtual Box** can use **qcow2** formatted images, and has a very close relationship with **qemu**.

In this course we recommend using (and will use in labs) **virt-manager** to configure and run virtual machines. We will also give instructions for how to run them using **qemu** command line utilities.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Image Formats

- Two main Formats:
  1. **cmd**: Simplest
  2. **qcow2**: Copy On Write 2
- Can support many other formats
- Can convert between formats:

```
$ qemu-img convert -O vmdk myvm.qcow2 myvm.vmdk
```

**QEMU** supports many formats for disk image files. However, only two are used primarily, with the rest being available for historical reasons and for conversion utilities. These two main formats are:

- **cmd** (default)  
This is the simplest and easiest to export to other non-**QEMU** emulators. Empty sectors do not take up space.
- **qcow2**  
**COW** stands for **C**opy **O**n **W**rite. There are many options. See **man qemu-img**.

To get a list of supported formats:

```
$ qemu-img --help | grep formats:
```

```
Supported formats: blkdebug blklogwrites blkreplay blkverify copy-on-read file ftp ftps gluster
↪ host_cdrom host_device http https iscsi iser luks nbd null-aio null-co nvme qcow2 quorum raw rbd ssh
↪ throttle vhdx vmdk vpc
```

Note in particular:

- **vdi**: Used by **Oracle Virtual Box**
- **vmdk**: Used by **VMware**

Note that **qemu-img** can be used to translate between formats, e.g.:

```
$ qemu-img convert -O vmdk myvm.qcow2 myvm.vmdk
```

using default options. See the **man** page for full possibilities.

**For the exclusive use of LFS301 corp class taught 06 to 09 December**

## 28.7 KVM

# KVM and Linux

- Kernel-based Virtual Machine
- Requires hardware virtualization extensions
- Uses kernel for resources in a co-processing relationship
- Kernel emulates guest instructions
- Avi Kivity submitted, quickly merged in 2007
- Has paravirtualization extensions and support

**KVM** uses the **Linux** kernel for computing resources including memory management, scheduling, synchronization, and more. When running a virtual machine, **KVM** engages in a co-processing relationship with the **Linux** kernel.

In this format, **KVM** runs the virtual machine monitor within one or more of the CPUs, using VMX or SVM instructions. At the same time the **Linux** kernel is executing on the other CPUs.

The virtual machine monitor runs the guest, which is running at full hardware speed, until it executes an instruction that causes the virtual machine monitor to take over.

At this point the virtual machine monitor can use any **Linux** resource to emulate a guest instruction, restart the guest at the last instruction, or do something else.

By loading the **KVM** modules and starting a guest, you turn **Linux** into a hypervisor. The **Linux** personality is still there, but you also have the hardware virtual machine monitor. You can control the Virtual Machine using standard **Linux** resource and process control tools, such as **cgroups**, **nice**, **numactl**, and so on.

**KVM** first appeared (pre-merge) as part of a Windows Virtual Desktop product. At the time of its upstream merge in 2007, **KVM** required recent **x86\_64** processors. On **x86\_64** platforms, **KVM** is primarily (but not always) a driver for the processor's virtualization subsystem.

**KVM** appeared as a trio of **Linux** kernel modules in 2007. When paired with a modified version of **QEMU** (also provided at the same time) it created a hypervisor that used the **Linux** kernel for most of its run-time services.

Shortly after Avi Kivity, the author of **KVM**, submitted the source code to the **Linux** development community, Linus merged **KVM** into his **Linux** tree. This was surprising to a lot of folks.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Managing KVM

- Both Command Line and Graphical Interfaces
- Command line tools include:
  - **virt-\***
  - **qemu-\***
- Graphical interfaces include:
  - **virt-manager**
  - **kimchi**
  - **OpenStack**
  - **oVirt**
- We will concentrate on **virt-manager**:
  - Most stable GUI across distributions and versions

There are many low level commands for creating, converting, manipulating, deploying and maintaining virtual machine images. As you develop more expertise you will become practiced in using them. But for all basic operations, **virt-manager** will suffice and that is what we will use.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 28.8 Labs

### Exercise 28.1: Making Sure KVM is Properly Set up



#### Very Important

- The following labs are best run on a physical machine running **Linux** natively.
- It may be possible to run them within a Virtual Machine running under a hypervisor, such as **VMWare** or **Virtual Box**, or even **KVM**. However, this requires **nested virtualization** to be running properly.
- Whether or not this works depends on the particular hypervisor used, the underlying host operating system (i.e., **Windows**, **Mac OS** or **Linux**) as well as the particular variant, such as which **Linux** or **Windows** version as well as the particular kernel.
- Furthermore it also depends on your particular hardware. For example, we have found nested virtualization working with **VMWare** on various **x86\_64** machines but with **Oracle Virtual Box** only on some.
- If this works, performance will be poor as compared to running on native hardware, but that is not important for the simple demonstrative exercises we will do.
- Your mileage will vary! If it does not work we cannot be responsible for helping you trying to get it rolling.

- First check that you have hardware virtualization available and enabled:

```
$ grep -e vmx -e svm /proc/cpuinfo
```

where **vmx** is for **INTEL** CPUs and **svm** for **AMD**. If you do not see either one of these:

- If you are on a physical machine, maybe you can fix this. Reboot your machine and see if you can turn on virtualization in the BIOS settings. Note that some IT personnel may make this impossible for “security” reasons, so try to get that policy changed.
- You are on a virtual machine running under a hypervisor, and you do not have nested virtualization operable.

- If for either of these reasons, you do not have hardware virtualization, you **may** be able to run **virt-manager**, but with weak performance.
- You need all relevant packages installed on your system. One can work hard to construct an exact list. However, exact names and requirements change with time, and most enterprise distributions ship with all (or almost all) of the software you need.
- The easiest and best procedure is to run the script we have already supplied to you:

```
$./ready-for.sh --install LFS301
```

where we have done the hard work.

Alternatively, on **RPM** systems you can do some overkill with:

```
$ sudo dnf|zypper install kvm* qemu* libvirt*
```

It is not a large amount of storage space to do it this way.

On **Debian** package based systems including **Ubuntu** you will have to do the equivalent with your favorite package installing procedure.



#### Very Important

- Do not run **libvirtd** at the same time as another hypervisor as dire consequences are likely to arise. This can easily include crashing your system and doing damage to any virtual machines being used.

**For the exclusive use of LFS301 corp class taught 06 to 09 December**



- We recommend both **stopping** and **disabling** your other hypervisor as in:

```
$ sudo systemctl stop vmware
$ sudo systemctl disable vmware
```

or

```
$ sudo systemctl stop vboxdrv
$ sudo systemctl disable vboxdrv
```

## Exercise 28.2: Using virt-manager with KVM to Install a Virtual Machine and Run it

In this exercise we will use pre-built **iso** images built by **TinyCoreLinux** (<http://www.tinycorelinux.net>) because they are cooked up very nicely and are quite small.

If you would like, you can substitute any installation **iso** image for another **Linux** distribution, such as **Debian**, **CentOS**, **Ubuntu**, **Fedora**, **openSUSE** etc. The basic steps will be identical and only differ when you get to the installation phase for building your new VM; which is no different than building any fresh installation on an actual physical machine.

We will give step-by-step instructions with screen capture images; If you feel confident, please try to just launch **virt-manager** and see if you can work your way through the necessary steps, as the **GUI** is reasonably clearly constructed.

1. Make sure **libvirtd** is running and start **virt-manager** by typing:

```
$ sudo systemctl start libvirtd
$ sudo virt-manager
```

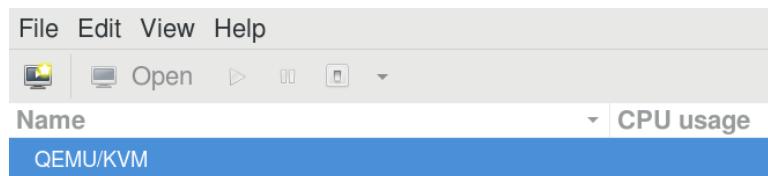


Figure 28.5: Starting virt-manager

2. Click on File->Create New Machine :

For the exclusive use of LFS301 corp class taught 06 to 09 December

Figure 28.6: Creating a Virtual Machine with `virt-manager`

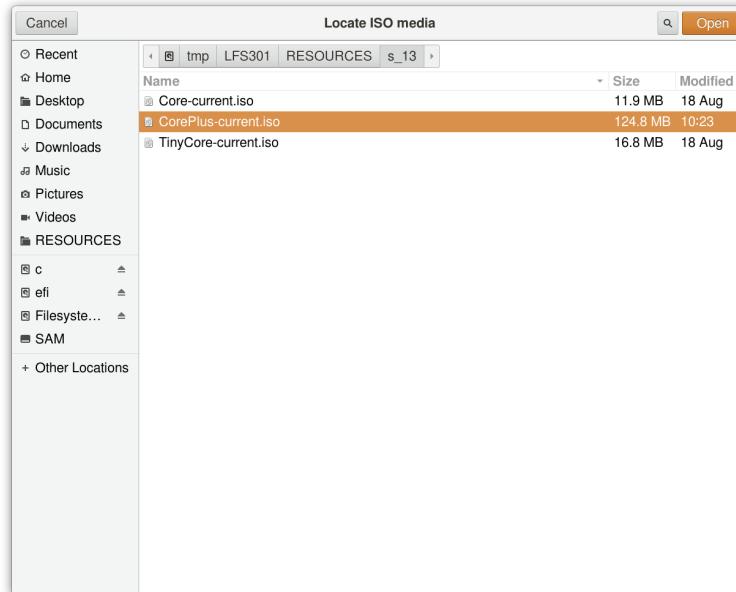
3. We have included three different **iso** install images from **TinyCoreLinux** in the `RESOURCES/s_28` directory:

```
Core-current.iso
CorePlus-current.iso
TinyCore-current.iso
```

(You can check and see if there are newer versions upstream at <http://www.tinycorelinux.net> but these should be fine.)

**CorePlus-current** is largest and robust and we will use this as it will install with full graphics. The others are considerably quicker to use, however.

Navigate through your file system and pick the desired image:

Figure 28.7: Selecting the TinyCoreLinux iso image in `virt-manager`

4. Next you have to request the amount of memory and number of **CPUs** or **cores** to use. These images are pretty minimal. A choice of 256 MB is more than enough; you may have fun seeing how low you can go!

For the exclusive use of LFS301 corp class taught 06 to 09 December

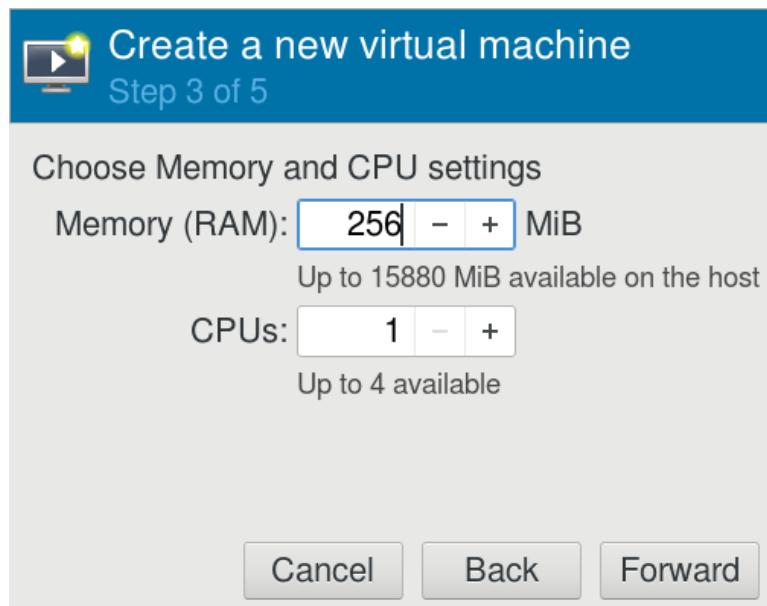


Figure 28.8: Configuring Memory and CPUS in virt-manager

5. Next you have to configure the location and size of the VM that is being created. You actually need very little for **TinyCoreLinux**, but the GUI will not let you choose less than 0.1 GB (about 100 MB.) (From the command line it is easy to configure less space.)

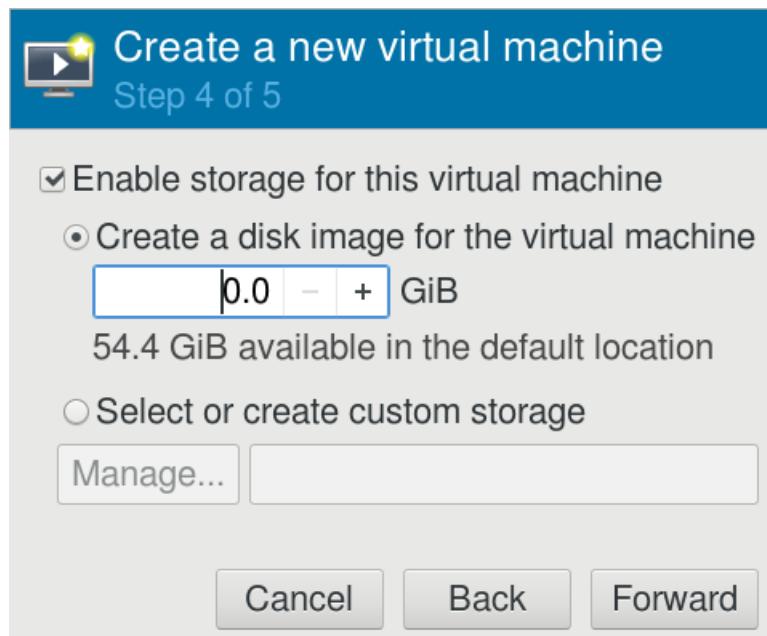


Figure 28.9: Configuring Disk Storage in virt-manager

If you do not click on Select or Create custom storage your image will be placed in `/var/lib/libvirt/images`. Since images can be quite large, you might want to configure to put it elsewhere. Or you can replace the images directory in `/var/lib/libvirt` with a symbolic link to somewhere else, as in:

```
$ cd /var/lib/libvirt
$ sudo mv images images_ORIGINAL
$ sudo mkdir /tmp/images
$ sudo ln -s /tmp/images images
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

(You probably want a different location for the images files than `/tmp`, but you get the idea.)

6. You are now ready to begin installation of your own VM from the **TinyCoreLinux** installation disk:

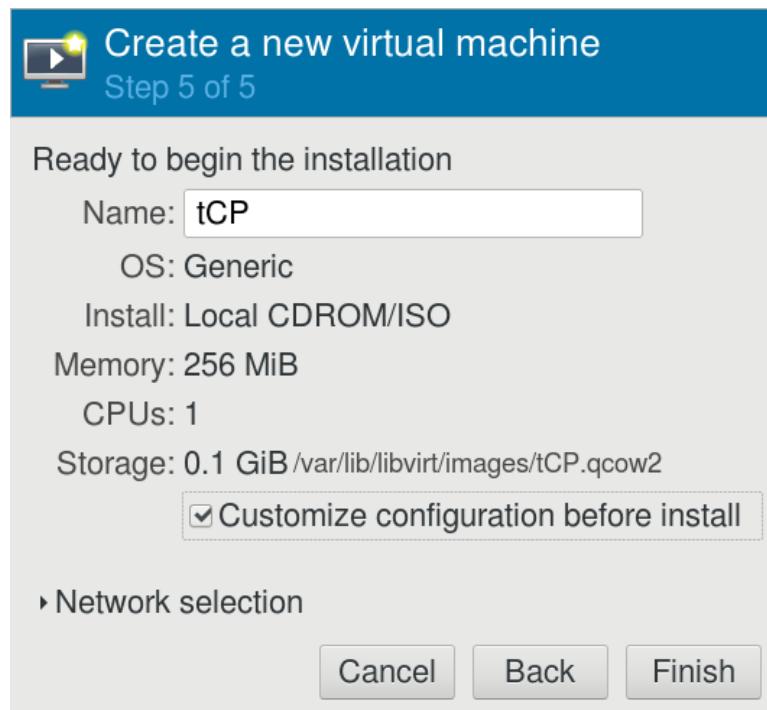


Figure 28.10: Beginning the VM installation in virt-manager



#### Please Note

We recommend clicking on **Customize** configuration before install. While you may want to make other changes, the mouse pointer is configured by default to be a **PS2** device; it is better to add a **USB tablet** input pointer.

Do this by clicking on `Add Hardware` on the next screen and then:

For the exclusive use of LFS301 corp class taught 06 to 09 December

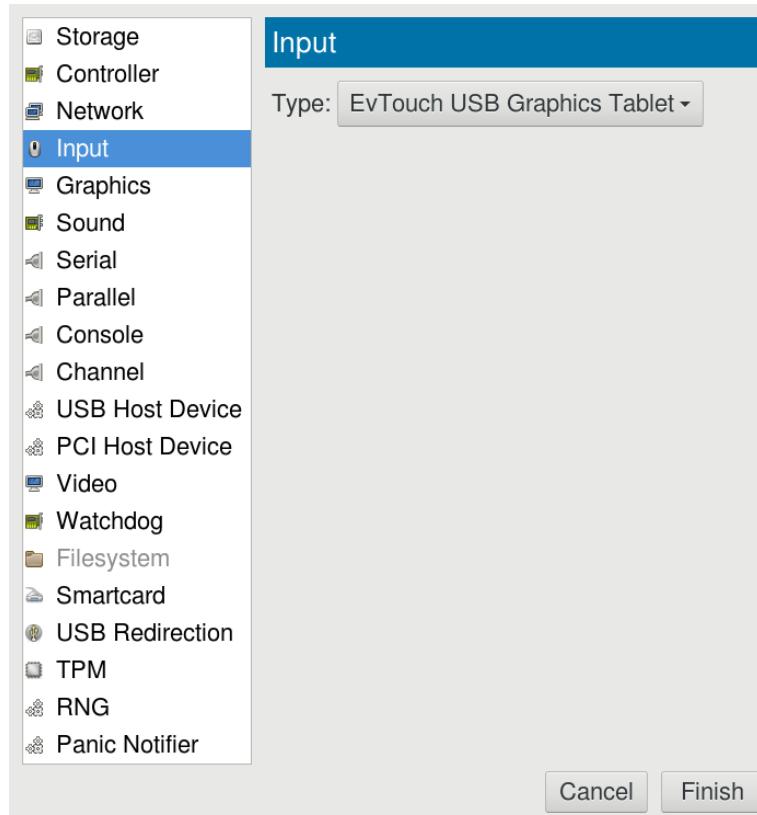


Figure 28.11: Adding an Input Device to the VM in virt-manager

7. Finally, we begin the installation:

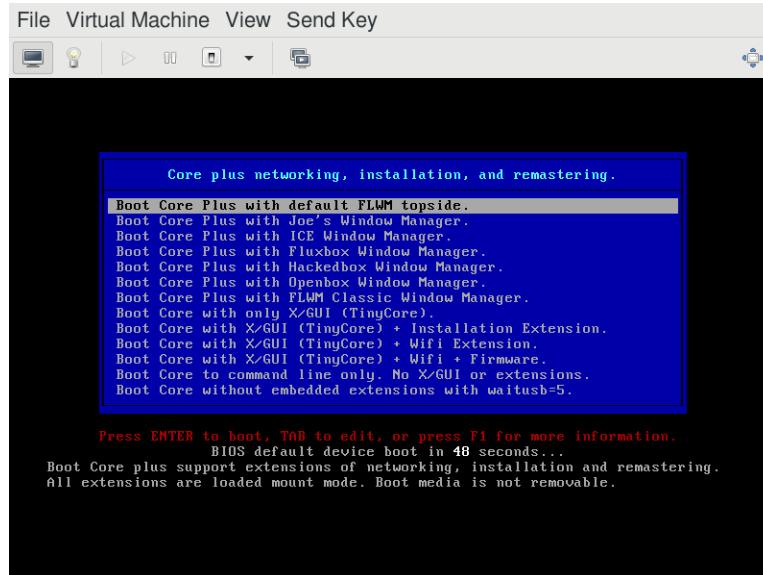


Figure 28.12: Booting into the Installation Media with virt-manager

You can make other choices for the graphical interface, here we just choose the first one, the default, and hit return.

8. This will take a while and eventually you will see the following screen:

For the exclusive use of LFS301 corp class taught 06 to 09 December

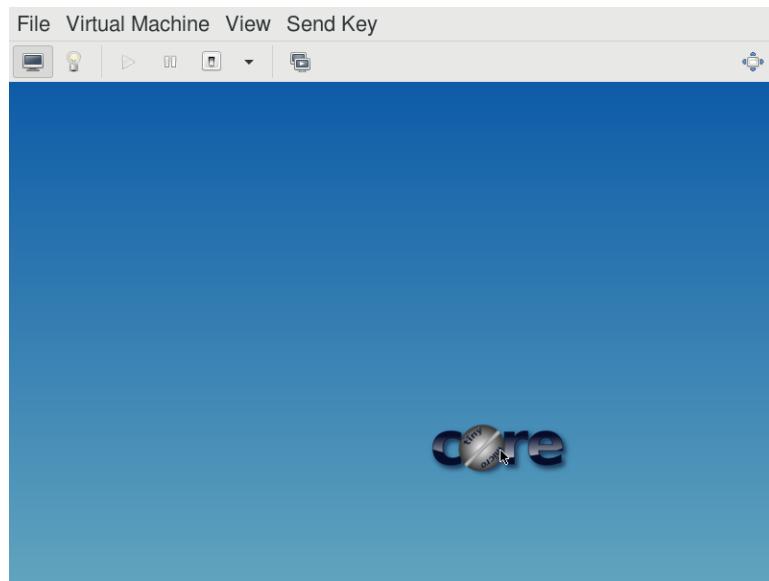


Figure 28.13: First TinyCoreLinux Screen

It is not obvious what to do here, but you need to see the icons at the bottom so you should resize and make the screen taller:

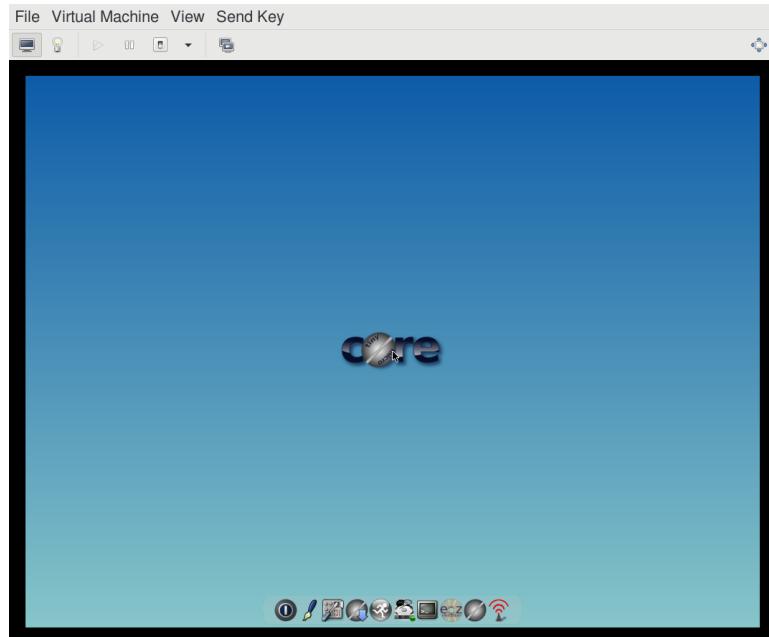


Figure 28.14: First TinyCoreLinux Screen Resized

9. Click on the terminal icon (or right click on the background and open up a terminal. Note the font is microscopic unfortunately. Then type

```
tc-install
```

in the window.

For the exclusive use of LFS301 corp class taught 06 to 09 December

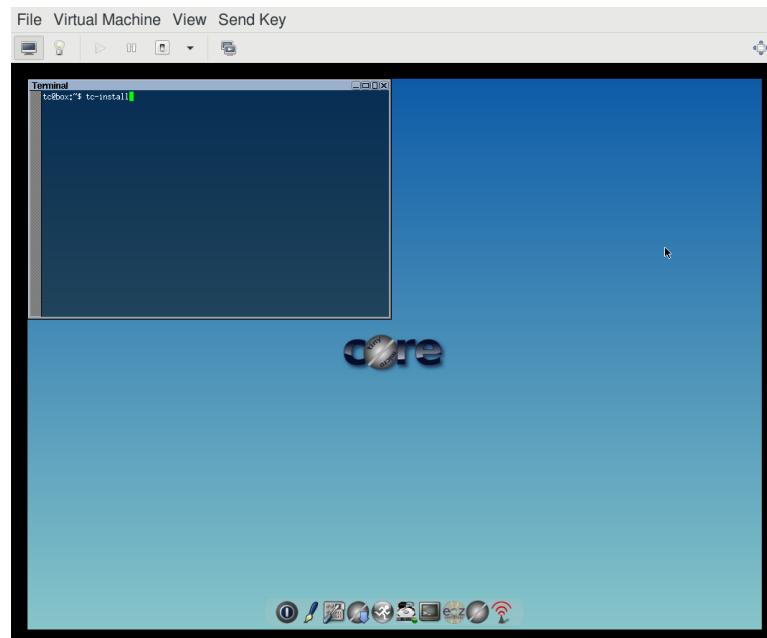


Figure 28.15: Running tc-install

10. Select Whole disk and sda and click the forward arrow:

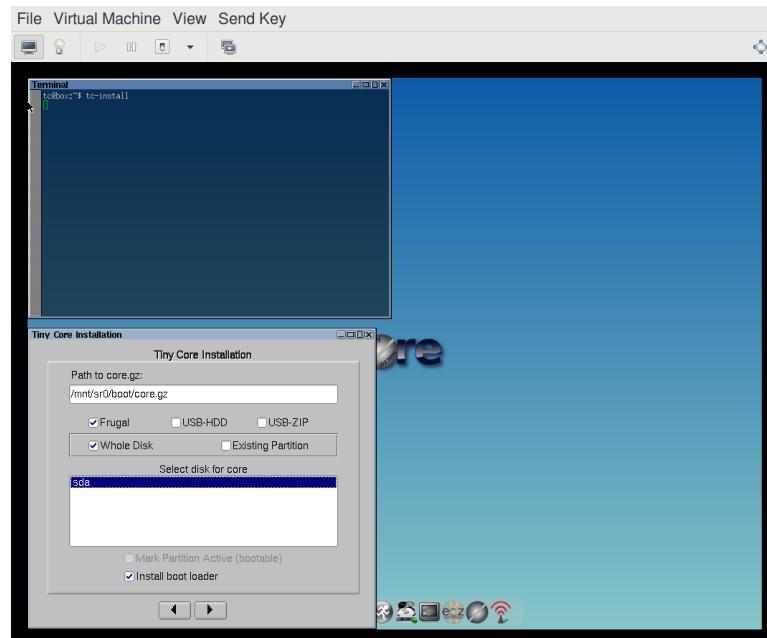


Figure 28.16: Selecting Disk in VM

11. Things will crank for a while and each step will be reflected in the output window.

For the exclusive use of LFS301 corp class taught 06 to 09 December

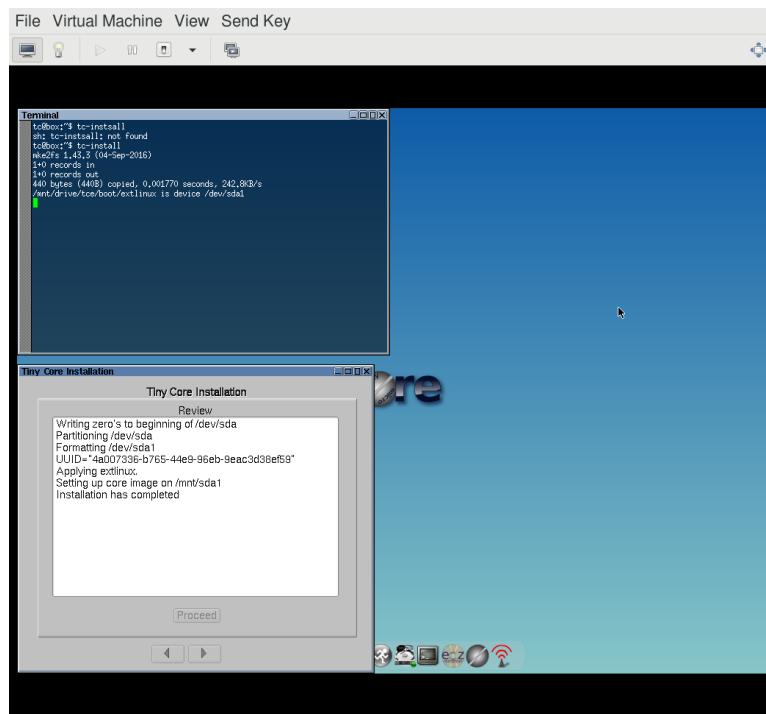


Figure 28.17: Finishing Installation in virt-manager

When installation is complete you can go to the File menu and shut down the virtual machine.

12. Start up **virt-manager** again (if you have killed it) and you should now see something like:

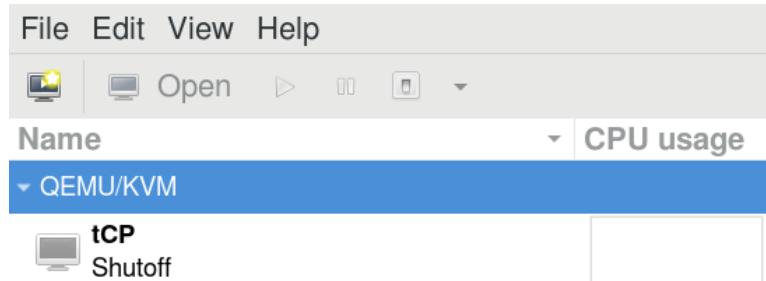


Figure 28.18: Running the new VM in virt-manager

Right click on the VM and open and run. Your new virtual machine should be up and running! (If you get confused and  
**For the exclusive use of LFS301 corp class taught 06 to 09 December**

think you are running the original install image, you can verify it is not that by noting there is no **tc-install** program in the new disk image.

## Exercise 28.3: Extra Credit: Doing from the command line

From the command line:

```
$ sudo qemu-img create -f qcow2 /var/lib/libvirtd/myimg.qcow2 24M
$ sudo qemu-system-x86_64 -hda /var/lib/libvirtd/myimg.qcow2 \
 -cdrom /teaching/LFCW/RESOURCES/LFS301/CorePlus-current.iso -usbdevice tablet
```



### Building QEMU from Source

**RHEL/CentOS 8** does not ship with **qemu-system-\*** binaries. It expects you to use the **kvm-qemu** software we have used in the previous exercises.

On these distributions, or on any system which does not provide these programs you can always compile from source; you may want to do this for various reasons even if your distribution provides the binary in order to get the latest version.

Note that this requires about 3 GB of storage during the build phase and takes quite a bit of time to accomplish. You may also be missing a number of development software packages and have to do some work to get things working. (These might include **ninja\*** and **libpulseaudio-devel**.)

The procedure for doing this is:

1. Get the QEMU source code:

```
$ git clone https://git.qemu.org/git/qemu.git
$ cd qemu
$ git submodule init
$ git submodule update --recursive
```

2. Configure QEMU to only build the `x86_64` version of `qemu-system`; building only the architecture we plan to use.

```
$./configure --target-list=x86_64-softmmu
```

3. Build QEMU as a non-root user.

```
$ make -j $(nproc)
```

4. Install QEMU into `/usr/local/bin/` as root.

```
$ sudo make install
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 29

# Containers Overview



|      |                                          |     |
|------|------------------------------------------|-----|
| 29.1 | Containers . . . . .                     | 354 |
| 29.2 | Application Virtualization . . . . .     | 355 |
| 29.3 | Containers vs Virtual Machines . . . . . | 356 |
| 29.4 | Docker . . . . .                         | 357 |
| 29.5 | Docker Commands . . . . .                | 359 |
| 29.6 | Podman . . . . .                         | 360 |
| 29.7 | Labs . . . . .                           | 361 |

## 29.1 Containers

# Container Basics

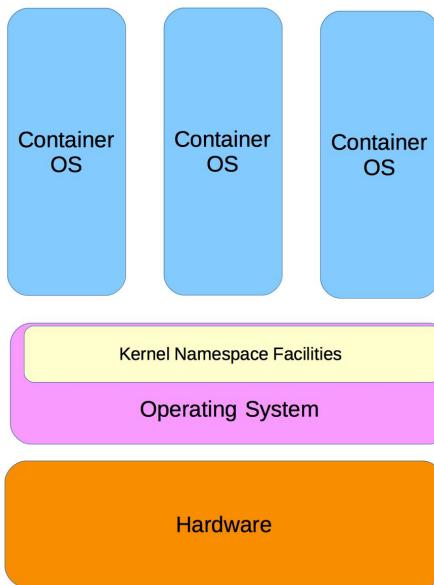


Figure 29.1: Containers

Further integration between the hypervisor and the **Linux** kernel allowed the creation of operating system level virtual machines or **Containers**. Containers share many facilities in the **Linux** kernel and make use of some recent kernel additions such as **namespaces** and **cgroups**. Containers are very light weight and reduce the overhead associated with having whole virtual machines.

The first flavor of containers was the **OS container**. This type of container runs an image of an operating system with the ability to run **init** processes and spawn multiple applications.

**LXC (Linux Containers)** is one example.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 29.2 Application Virtualization

# Application Virtualization

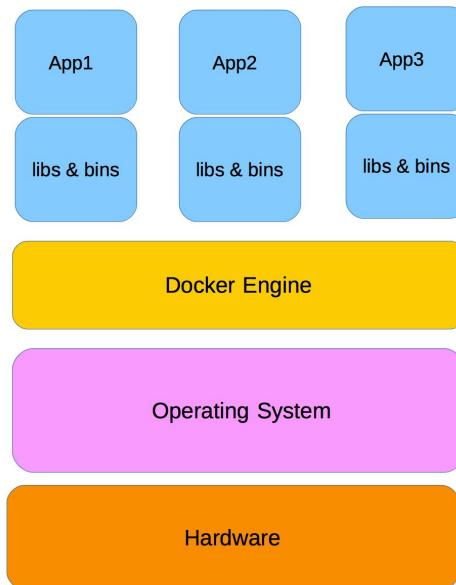


Figure 29.2: Application Virtualization

In an effort to further reduce the overhead associated with virtual machines, **application virtualization** is rising in popularity. Application virtualization runs one application for each container. Many single application containers are typically initialized on a single machine. Using smaller components creates a greater flexibility and reduces overhead normally associated with virtualization.

Docker is one such project.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 29.3 Containers vs Virtual Machines

# Containers vs Virtual Machines

- Running Processes and Services
  - VM runs a complete operating system and can run many services and applications
  - Container usually runs one thing
- VM uses more resources than a container
- Multiple containers share one OS kernel, each VM has its own
- Containers are more portable
- Containers can be run inside a VM
- Containers are harder to secure
- Containers usually start faster
- However, Virtual Machines are still often the best solution

Both Virtual Machines and Containers satisfy important needs. For a while each facility had its day in the sun and was seen as the way to go for almost everything.

Both have long histories:

- Main frame computers have had software partitioning and virtual machines for decades, in rather specialized ways.
- Operating systems have had **chroot** and **BSD Jail** implementations for many years, that share a basic isolation motivation with containers.

If a number of different services and applications need to be tightly integrated, a virtual machine functioning as a service may be the best solution.

If the applications being run were written expecting a complete operating system environment with a wide range of services and other libraries and applications, virtual machines may be best.

Scaling workloads is different for containers and virtual machines. **Orchestration** systems such as **Kubernetes** or **Mesos** can decide on the proper quantity of containers needed, do load balancing, replicate images and remove them etc as needed.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 29.4 Docker

# Docker

**Docker** is an application-level virtualization using many individual images to build up the necessary services to support the target application. These images are packaged into containers.

- Images are components in containers
- Images may contain
  - application code
  - runtime libraries
  - system tools
  - ... or just about anything required for an application
- Images may reside on a **Docker Hub** or a **registry** server

**Docker** has extensive documentation at <https://docs.docker.com>

The **Docker** website <https://www.docker.com> has documentation, tutorials and training information.

One of the most compelling features of **Docker** is an application can be packaged up with all of its dependent code and services and deployed as a single unit with the minimum of overhead. This deployment can be easily repeated as often as desired. This reduces the requirement of building up a server with layered services to support the end application.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Docker Steps

Starting up a **Docker** containerized application may include only a few steps.

- Install the **Docker** service package with your favorite tool.
- Start the **Docker** service.
- Search for an appropriate image from **Docker Hub** or your private repository.
- Pull the image.
- Run the image
- and finally test the application.

The steps detailed above are just a very minimal example of testing a **Docker** application.

There are of course many, many options that may be used including functions that create an image, set system variables or configuration parameters, and then store the result as a new image. In some cases, a writable image is required, rather than a non-writable one.

Most of the **Docker** commands have individual **man** pages. Examples include: `docker(1)`, `docker-search(1)`, `docker-pull(1)`, `docker-create(1)`, and `docker-run(1)`.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 29.5 Docker Commands

# docker Command

- The **docker** command has more than 40 sub-commands
  - Some with 50 or more options
- docker <command> --help for details
  - run
  - create
  - exec
  - ps
  - images
  - network
  - rm
  - rmi
  - save
  - search
  - start
  - stop
  - top
  - update
  - volume
- etc.

There are many **docker** sub-commands and they tend to be somewhat self-documenting. Often confused are `run`, `create` and `exec`. The `ps` command will list running containers, or all containers if you include the `-all` option.

`run` will start a new container and execute a command within. Common options are `-t` to attach to a tty and `-d` to run the container in the background.

The `create` command creates a container. It has many options for configuring container settings and attachments.

If the container is already running, and you want to execute something inside of it you can use the `exec` command. It also accepts the `-t` `-d` options.

The `images` command will show images in various outputs. The `rmi` command will remove images and delete un-tagged parents by default.

You can also leverage shell functions to operate upon all containers. For example to remove all stopped containers:  
`docker rm $(docker ps -a -q)`.

## 29.6 Podman

# Podman

- RHEL8/CentOS8 have replaced pure **docker** with **Podman**
- **Podman** uses a child/parent forking model for container creation and management
- **Docker** uses a server/client model with a daemon running in background for management
- Promised benefits:
  - Better security
  - Less overhead
- \$ sudo dnf install podman podman-docker
- Emulation layer enables backwards compatibility with **docker** commands

We will show how to run the lab exercise on **RHEL/CentOS 8** using the **docker** emulation layer. Other distributions have been adding **podman** to their packaging systems.

On **Ubuntu 20.04** the official repositories already include **podman** so you can just do:

```
$ sudo apt-get update
$ sudo apt-get install podman
```

On **Ubuntu 18.04** you have to first do:

```
$ sudo apt-get update
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository ppa:projectatomic/ppa
```

## 29.7 Labs

### Exercise 29.1: Install and test Apache (httpd) as a Docker application.

#### Overview

In this exercise, we will install, run and test the **docker** package, and follow with getting and deploying **httpd**, the **Apache** web server container.



#### On RedHat, CentOS, or Fedora

Please note that **RHEL/CentOS 8** and recent **Fedora** distributions utilized **podman**. On these systems **docker** is supported by a backwards compatibility layer:

```
$ sudo dnf install podman podman-docker
```

1. Make sure **Docker** is installed (or emulated with **podman**.) Pick the right command for your distribution:

```
$ sudo yum install docker # RHEL/CentOS 7
$ sudo dnf install podman podman-docker # RHEL/CentOS 8, Fedora
$ sudo apt-get install docker.io # Ubuntu, Debian
$ sudo zypper install docker # openSUSE
```



#### Reinstall Docker?

- If you get strange errors at later points in the exercise you might find it useful to **reinstall docker**. We have observed cases (for example, with **RHEL 7**) where **docker** configurations were broken, after a system upgrade,

2. Start the **docker** service:



#### Very Important

You can skip to the next step on **podman**-based systems as there is no **docker** service to start!

```
$ sudo systemctl start docker
```

You may want to verify that it is running properly with `systemctl status docker`:

```
~ student@ubuntu:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
 Loaded: loaded (/lib/systemd/system/docker.service; disabled; vendor preset: enabled)
 Active: active (running) since Thu 2021-03-25 11:48:49 PDT; 8min ago
 TriggeredBy: ● docker.socket
 Docs: https://docs.docker.com
 Main PID: 3063 (dockerd)
 Tasks: 12
 Memory: 125.7M
 CGroup: /system.slice/docker.service
 └─3063 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Mar 25 11:48:48 ubuntu dockerd[3063]: time="2021-03-25T11:48:48.994073149-07:00" level=warning msg="Your kernel does not support cgroup rt runtime"
Mar 25 11:48:48 ubuntu dockerd[3063]: time="2021-03-25T11:48:48.994324640-07:00" level=warning msg="Your kernel does not support cgroup blkio weight"
Mar 25 11:48:48 ubuntu dockerd[3063]: time="2021-03-25T11:48:48.99465112-07:00" level=warning msg="Your kernel does not support cgroup blkio weight"
Mar 25 11:48:48 ubuntu dockerd[3063]: time="2021-03-25T11:48:48.995099733-07:00" level=info msg="Loading containers: start."
Mar 25 11:48:49 ubuntu dockerd[3063]: time="2021-03-25T11:48:49.337342379-07:00" level=info msg="Default bridge (dockero) is assigned with an IP address"
Mar 25 11:48:49 ubuntu dockerd[3063]: time="2021-03-25T11:48:49.633735493-07:00" level=info msg="Loading containers: done."
Mar 25 11:48:49 ubuntu dockerd[3063]: time="2021-03-25T11:48:49.667539540-07:00" level=info msg="Docker daemon" commit=afacb8b7f0 graphdriver(s)=devicemapper
Mar 25 11:48:49 ubuntu dockerd[3063]: time="2021-03-25T11:48:49.667912940-07:00" level=info msg="Daemon has completed initialization"
Mar 25 11:48:49 ubuntu systemd[1]: Started Docker Application Container Engine.
Mar 25 11:48:49 ubuntu dockerd[3063]: time="2021-03-25T11:48:49.691798667-07:00" level=info msg="API listen on /run/docker.sock"
student@ubuntu:~$
```

Figure 29.3: Checking docker status

For the exclusive use of LFS301 corp class taught 06 to 09 December

If you see anything indicating failure you should inspect `/var/log/messages` or whatever other logging file you have on your system for clues. If you are running a standard distribution kernel you should be fine, but if you are running a custom **Linux** kernel, it is likely you have to select the proper configuration options, especially as regards to networking. This is too complicated to go into here, so please stay with a distribution supplied kernel unless you want a challenging exercise!

### 3. Search for the **httpd** container, with

```
$ sudo docker search apache
```

| NAME                             | DESCRIPTION                                     | STARS | OFFICIAL | AUTOMATED |
|----------------------------------|-------------------------------------------------|-------|----------|-----------|
| httpd                            | The Apache HTTP Server Project                  | 3424  | [OK]     |           |
| tomcat                           | Apache Tomcat is an open source implementati... | 2979  | [OK]     |           |
| cassandra                        | Apache Cassandra is an open-source distribut... | 1251  | [OK]     |           |
| maven                            | Apache Maven is a software project managemen... | 1174  | [OK]     |           |
| zookeeper                        | Apache ZooKeeper is an open-source server wh... | 1041  | [OK]     |           |
| solr                             | Solr is the popular, blazing-fast, open sour... | 819   | [OK]     |           |
| apache/airflow                   | Apache Airflow                                  | 226   |          |           |
| apache/nifi                      | Unofficial convenience binaries and Docker i... | 206   | [OK]     |           |
| eboraas/apache-php               | PHP on Apache (with SSL/TLS support), built ... | 144   | [OK]     |           |
| apache/zeppelin                  | Apache Zeppelin                                 | 144   | [OK]     |           |
| eboraas/apache                   | Apache (with SSL/TLS support), built on Debi... | 92    | [OK]     |           |
| apacheignite/ignite              | Apache Ignite - Distributed Database            | 76    | [OK]     |           |
| nimmis/apache-php5               | This is docker images of Ubuntu 14.04 LTS wi... | 69    | [OK]     |           |
| bitnami/apache                   | Bitnami Apache Docker Image                     | 67    | [OK]     |           |
| apachepulsar/pulsar              | Apache Pulsar - Distributed pub/sub messagin... | 34    |          |           |
| linuxserver/apache               | An Apache container, brought to you by Linux... | 27    |          |           |
| apache/nutch                     | Apache Nutch                                    | 23    | [OK]     |           |
| antage/apache2-php5              | Docker image for running Apache 2.x with PHP... | 23    | [OK]     |           |
| webdevops/apache                 | Apache container                                | 15    | [OK]     |           |
| newdeveloper/apache-php          | apache-php7.2                                   | 8     |          |           |
| newdeveloper/apache-php-composer | apache-php-composer                             | 7     |          |           |
| lephare/apache                   | Apache container                                | 6     | [OK]     |           |
| secoresearch/apache-varnish      | Apache+PHP+Varnish5.0                           | 2     | [OK]     |           |
| apache/arrow-dev                 | Apache Arrow convenience images for developm... | 1     |          |           |
| jelastis/apachephp               | An image of the Apache PHP application serve... | 0     |          |           |

Figure 29.4: Using docker search

(You could have used **httpd** instead of **apache** in the above command with very similar results.)

From now on we will not show detailed output since if you have gotten this far, things should be fine.

### 4. Retrieve the container:

```
$ sudo docker pull docker.io/httpd
```

This may take a couple of minutes while all the components download.

### 5. List the installed containers:

```
$ sudo docker images
```

### 6. List the components associated with the images.

```
$ sudo docker images --all
```

### 7. Start the **httpd** docker container. The terminal will appear to hang as it is now connected to the **httpd** daemon.

```
c7:/tmp>sudo docker run httpd
```

```
AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
using 172.17.0.2. Set the 'ServerName' directive globally to suppress this message
.....
```

### 8. You can open a graphical web browser pointing to the IP address in the above output. (Do not use the address shown in the output above!)

Or you can use a text-based browser (especially if you are not in a graphical environment) by opening up a new terminal window (do not kill the one in which the **docker httpd** container is running!) and doing one of the following commands:

For the exclusive use of LFS301 corp class taught 06 to 09 December

```
$ lynx http://172.17.0.2
$ w3m http://172.17.0.2
$ elinks http://172.17.0.2
```

using whichever text-based browser is installed on your system.

9. Stop the container and **docker** service and clean up.

```
c7:/tmp>sudo docker ps
```

|   | CONTAINER ID | IMAGE | COMMAND            | CREATED        | STATUS        | PORTS  | NAMES         |
|---|--------------|-------|--------------------|----------------|---------------|--------|---------------|
| 1 | b936b0afeb23 | httpd | "httpd-foreground" | 41 seconds ago | Up 40 seconds | 80/tcp | boring_turing |

```
c7:/tmp>sudo docker stop b936b0afeb23
```

|   |              |
|---|--------------|
| 1 | b936b0afeb23 |
|---|--------------|

10. This will leave images and their associated storage under either `/var/lib/docker` or `/var/lib/containers` depending on your particular system and distribution. If you do not need to reuse them you can clean up with:

```
c7:/tmp>sudo docker rmi -f docker.io/httpd
Untagged: docker.io/httpd:latest
Untagged: docker.io/httpd@sha256:cf774f082e92e582d02acdb76dc84e61dcf5394a90f99119d1ae39bcecbff075
Deleted: sha256:cf6b6d2e846326d2e49e12961ee0f63d8b5386980b5d3a11b8283151602fa756
```

and on some systems you may also need to do: Deleted Containers:

```
c7:/tmp>sudo docker system prune -a
....
```

On non-**podman** systems you may also want to do:

```
c7:/tmp>sudo systemctl stop docker
```

---

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 30

# User Account Management



|       |                                             |     |
|-------|---------------------------------------------|-----|
| 30.1  | User Accounts . . . . .                     | 366 |
| 30.2  | Management of User Accounts . . . . .       | 368 |
| 30.3  | Locked Accounts . . . . .                   | 370 |
| 30.4  | Passwords . . . . .                         | 371 |
| 30.5  | /etc/shadow   . . . . .                     | 372 |
| 30.6  | Password Management . . . . .               | 374 |
| 30.7  | Password Aging . . . . .                    | 375 |
| 30.8  | Restricted Shells and Accounts ** . . . . . | 376 |
| 30.9  | The root Account . . . . .                  | 378 |
| 30.10 | SSH . . . . .                               | 379 |
| 30.11 | Labs . . . . .                              | 382 |



### Please Note

\*\* These sections may be considered in part or in whole as optional. They contain either background reference material, specialized topics, or advanced subjects. The instructor may choose to cover or not cover them depending on classroom experience and time constraints.

## 30.1 User Accounts

# Purpose of User Accounts

- Providing each user with their own individualized private space.
- Creating particular user accounts for specific dedicated purposes.
- Distinguishing privileges among users.

Linux systems provide a **multi-user** environment which permits people and processes to have separate simultaneous working environments.

One special user account is for the **root** user, who is able to do **anything** on the system. To avoid making costly mistakes, and for security reasons, the root account should only be used when absolutely necessary.

Normal user accounts are for people who will work on the system. Some user accounts (like the **daemon** account) exist for the purpose of allowing processes to run as a user other than root.

We will also have a discussion of **group** management where subsets of the users on the system can share files, privileges etc. according to common interests.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Attributes of a User Account

From `/etc/passwd`:

```
....
beav:x:1000:1000:Theodore Cleaver:/home/beav:/bin/bash
warden:x:1001:1001:Ward Cleaver:/home/warden:/bin/bash
dobie:x:1002:1002:Dobie Gillis:/home/dobie:/bin/bash
....
```

- User name
- User password
- User identification number (**UID**)
- Group identification number (**GID**)
- Comment or **GECOS** information
- Home directory
- Login shell

Each user on the system has a corresponding line in the `/etc/passwd` file that describes their basic account attributes. (We will talk about passwords as well as this file later). The seven elements here are:

1. User name  
The unique name assigned to each user.
2. User password  
The password assigned to each user.
3. User identification number (**UID**)  
A unique number assigned to the user account. The **UID** is used by the system for a variety of purposes, including a determination of user privileges and activity tracking.
4. Group identification number (**GID**)  
Indicates the primary, principal, or default **group** of the user.
5. Comment or **GECOS** information  
A defined method to use the comment field for contact information (full name, email, office, contact number). (Do not worry about what **GECOS** means, it is a very old term.)
6. Home directory  
For most users, this is a unique directory that offers a working area for the user. Normally, this directory is owned by the user, and except for **root** will be found on the system somewhere under `/home`.
7. Login shell  
Normally, this is a shell program such as `/bin/bash` or `/bin/csh`. Sometimes, however, an alternative program is referenced here for special cases. In general, this field will accept any executable.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 30.2 Management of User Accounts

### Creating User Accounts with useradd

- **useradd** allows for default operation

```
$ sudo useradd dexter
```

- Creates account dexter
- Uses defaults for user and group id, home directory, and shell

- Override defaults by using options to **useradd**

```
$ sudo useradd -s /bin/csh -m -k /etc/skel -c "Bullwinkle J Moose" bmoose
```

- Specifies shell, skel directory, comment field using options

```
$ sudo useradd dexter
```

causes the following steps to execute:

- The next available UID greater than UID\_MIN (specified in `/etc/login.defs`) by default is assigned as dexter's UID.
- A group called dexter with a GID=UID is also created and assigned as dexter's primary group.
- A home directory `/home/dexter` is created and owned by dexter.
- dexter's login shell will be `/bin/bash`.
- The contents of `/etc/skel` is copied to `/home/dexter`. By default, `/etc/skel` includes startup files for **bash** and for the **X Window** system.
- An entry of !! is placed in the password field of the `/etc/shadow` file for dexter's entry, thus requiring the administrator to assign a password for the account to be usable.

The defaults can easily be overruled by using options to **useradd** as in:

```
$ sudo useradd -s /bin/csh -m -k /etc/skel -c "Bullwinkle J Moose" bmoose
```

where explicit non-default values have been given for some of the user attributes.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Modifying and Deleting User Accounts

- The root user can delete user accounts with **userdel**

```
$ sudo userdel morgan
```

Deletes the `morgan` user account, but does not remove her home directory

- User accounts can be modified with **usermod**

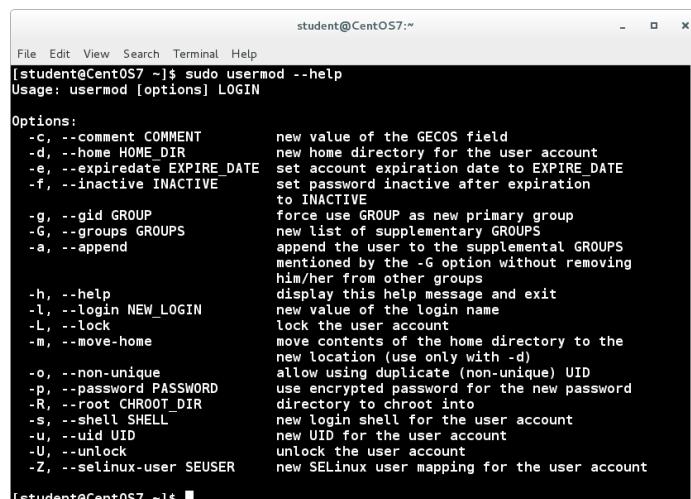
```
$ sudo usermod -L dexter
```

Locks the account for `dexter` so he cannot login

```
$ sudo userdel morgan
```

All references to the user `morgan` will be erased from `/etc/passwd`, `/etc/shadow`, and `/etc/group`. While this removes the account, it does not delete the home directory (usually `/home/morgan`) in case the account may be re-established later. If the `-r` option is given to `userdel`, the home directory will also be obliterated. However, all other files on the system owned by the removed user will remain.

**usermod** can be used to change characteristics of a user account such as group memberships, home directory, login name, password, default shell, user id etc. Usage is pretty straightforward. Note **usermod** will take care of any modifications to files in the `/etc` directory as necessary.



A screenshot of a terminal window titled "student@CentOS7:~". The window displays the usage information for the `usermod` command. The text is as follows:

```
[student@CentOS7 ~]$ sudo usermod --help
Usage: usermod [options] LOGIN

Options:
 -c, --comment COMMENT new value of the GECOS field
 -d, --home HOME_DIR new home directory for the user account
 -e, --expiredate EXPIRE_DATE set account expiration date to EXPIRE_DATE
 -f, --inactive INACTIVE set password inactive after expiration
 to INACTIVE
 -g, --gid GROUP force use GROUP as new primary group
 -G, --groups GROUPS new list of supplementary GROUPS
 -a, --append append the user to the supplemental GROUPS
 mentioned by the -G option without removing
 him/her from other groups
 -h, --help display this help message and exit
 -l, --login NEW_LOGIN new value of the login name
 -L, --lock lock the user account
 -m, --move-home move contents of the home directory to the
 new location (use only with -d)
 -o, --non-unique allow using duplicate (non-unique) UID
 -p, --password PASSWORD use encrypted password for the new password
 -R, --root CHROOT_DIR directory to chroot into
 -s, --shell SHELL new login shell for the user account
 -u, --uid UID new UID for the user account
 -U, --unlock unlock the user account
 -Z, --selinux-user SEUSER new SELinux user mapping for the user account

[student@CentOS7 ~]$
```

Figure 30.1: Using `usermod`

For the exclusive use of LFS301 corp class taught 06 to 09 December

### 30.3 Locked Accounts

## Locked Accounts

- **Linux** ships with some system accounts that are **locked**:
  - Accounts such as bin, daemon, or sys
  - May run programs but are never used for login purposes
- Other locked accounts may be created for other purposes
  - Accounts used by major applications like a database
- Locked account has no valid password
  - Usually represented by "!!"
  - Fewer or greater than 100 characters for password
- Ensure that all locked accounts also have an invalid shell (generally `/sbin/nologin`)

**Linux** ships with some **locked** accounts, which means they can run programs, but can never login to the system and have no valid password associated with them. For example `/etc/passwd` has entries like:

```
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
```

The **nologin** shell returns the following if a locked user tries to login to the system:

```
This account is currently not available.
```

or whatever message may be stored in `/etc/nologin.txt`. Such locked accounts are created for special purposes, either by system services or applications; if you scan `/etc/passwd` for users with the **nologin** shell you can see who they are on your system. It is also possible to lock the account of a particular user as in:

```
$ sudo usermod -L dexter
```

which means the account stays on the system but logging in is impossible. Unlocking can be done with the `-U` option. A customary practice is to lock a user's account whenever they leave the organization or is on an extended leave of absence. Another way to lock an account is to use **chage** to change the expiration date of an account:

```
$ sudo chage -E 2014-09-11 morgan
```

The actual date is irrelevant as long as it is in the past.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 30.4 Passwords

### User IDs and /etc/passwd

```
beav:x:1000:1000:Theodore Cleaver:/home/beav:/bin/bash
rsquirrel:x:1001:1001:Rocket J Squirrel:/home/rsquirrel:/bin/bash
```

- `/etc/passwd` file contains one record (one line) for each user, each of which is a colon ( : ) separated list of fields:
  - username: user's unique name
  - password: either the hashed password (if `/etc/shadow` is not used) or a place holder ("x" when `/etc/shadow` is used)
  - UID: user identification number
  - GID: primary group identification number for the user
  - comment: comment area, usually the user's real name
  - home directory pathname for the user's home directory
  - shell: absolutely qualified name of the shell to invoke at login
- UIDs start at `UID_MIN = 1000`

The convention most **Linux** distributions have used is that any account with a **UID** less than 1000 is considered special and belongs to the system; normal user accounts start at 1000. The actual value is defined as `UID_MIN` and is defined in `/etc/login.defs`.

Historically, **Red Hat**-derived distributions used `UID_MIN=500`, not 1000, but beginning with **RHEL 7** the more common value of 1000 was adopted.

If a **UID** is not specified when using `useradd`, the system will incrementally assign user IDs starting at `UID_MIN`.

Additionally, each user gets a **Primary Group ID** which by default is the same number as the **UID**. These are sometimes called **User Private Groups** (UPG).

It is bad practice to edit `/etc/passwd`, `/etc/group` or `/etc/shadow` directly; either use appropriate utilities such as `usermod`.

## 30.5 /etc/shadow I

### Why Use /etc/shadow?

- Enables password aging on a per user basis
- Allows for greater security of hashed passwords

The default permissions of `/etc/passwd` are 644 (`-rw-r--r--`); anyone can read the file. This is unfortunately necessary because system programs and user applications need to read the information contained in the file. These system programs do not run as the user root and, in any event, only root may change the file.

Of particular concern are the hashed passwords themselves. If they appear in `/etc/passwd`, anyone may make a copy of the hashed passwords and then make use of utilities such as **Crack** and **John the Ripper** to guess the original cleartext passwords given the hashed password. This is a security risk!

`/etc/shadow` has permission settings of 400 (`-r-----`), which means that only root can access this file. This makes it more difficult for someone to collect the hashed passwords.

Unless there is a compelling good reason not to, you should use the `/etc/shadow` file.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## /etc/shadow II

- `/etc/shadow` file contains one record (one line) for each user
- Each record contains fields separated by colons ( : )
  - `username`: unique user name.
  - `password`: the hashed (sha512) value of the password
  - `lastchange`: days since Jan 1, 1970 that password was last changed
  - `mindays`: minimum days before password can be changed
  - `maxdays`: maximum days after which password must be changed
  - `warn`: days before password expires that the user is warned
  - `grace`: days after password expires that account is disabled
  - `expire`: date that account is/will be disabled
  - `reserved`: reserved field

`/etc/shadow` contains one record (one line) for each user as in:



### /etc/shadow

```
daemon:*:16141:0:99999:7:::
.....
beav:6iCZyCnBJH9rmq7P.$RYNm10Jg3wrhAtUnahBZ/mTMg.RzQE6iBXyqaXHvxxbK\
TYqj.d9wpoQFuRp7fPEE3hMK3W2gcIYhiXa9MIA9w1:16316:0:99999:7:::
```

The username in each record must exactly match that found in `/etc/passwd`, and also must appear in the identical order.

All dates are stored as the number of days since Jan. 1, 1970 (the **epoch** date).

The password hash is the string “\$6\$” followed by an eight character salt value, which is then followed by a \$ and an 88 character (sha512) password hash.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 30.6 Password Management

# Password Management

- Passwords can be changed with **passwd**
- Users can change their own password
- Root can change any user password

By default, the password choice is examined by **pam\_cracklib.so**, which furthers making good password choices.

A normal user changing their password:

```
$ passwd
1 Changing password for clyde
2 (current) UNIX password: <clyde's password>
3 New UNIX password: <clyde's-new-password>
4 Retype new UNIX password: <clyde's-new-password>
5 passwd: all authentication tokens updated successfully
```

Note that when root changes a user's password, root is not prompted for the current password:

```
$ sudo passwd kevin
1 New UNIX password: <kevin's-new-password>
2 Retype new UNIX password: <kevin's-password>
3 passwd: all authentication tokens updated successfully
```

Note that normal users will not be allowed to set bad passwords, such as ones that are too short, or based on dictionary words. However, root is allowed to do so.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 30.7 Password Aging

# Password Aging (chage)

- Managed through the use of chage

```
$ chage [-m mindays] [-M maxdays] [-d lastday] \
 [-I inactive] [-E expiredate] [-W warndays] user
```

- Examples:

```
$ sudo chage -l stephane
$ sudo chage -m 14 -M 30 kevlin
$ sudo chage -E 2012-4-1 isabelle
$ sudo chage -d 0 clyde
```

It is generally considered important to change passwords periodically. This limits the amount of time a cracked password can be useful to an intruder and also can be used to lock unused accounts. The downside is users can find this policy annoying and wind up writing down their ever-changing passwords and thus making them easier to steal. The utility that manages this is **chage**:

```
chage [-m mindays] [-M maxdays] [-d lastday] [-I inactive] [-E expiredate] [-W warndays] user
```

```
student@ubuntu:/etc/udev$ chage -l student
Last password change : Apr 14, 2017
Password expires : never
Password inactive : never
Account expires : never
Minimum number of days between password change : 0
Maximum number of days between password change : 99999
Number of days of warning before password expires : 7
```

Figure 30.2: Using chage

Only the root user can use **chage**. The one exception to this is that any user can run with the **-l** option to determine when their password or account is due to expire.

To force a user to change their password at their next login:

```
$ sudo chage -d 0 USERNAME
For the exclusive use of LFS301 corp class taught 06 to 09 December
```

## 30.8 Restricted Shells and Accounts \*\*

### Restricted Shell

Linux includes a **restricted shell**, which can be invoked as:

```
$ bash -r
```

- Prevents using **cd** to change directories
- Prevents setting the SHELL, ENV, or PATH environment variables
- Prohibits specifying path or command names containing **/**
- Restricts redirection of output and/or input
- **Restricted Accounts** can also be enabled by creating a symlink to **/bin/bash** named **/bin/rbash** and using in **/etc/passwd** as we will discuss next



#### rbash is not secure!

It is actually very easy to avoid the restrictions and modern techniques such as the use of **SELinux** are much more robust. We discuss only if you encounter the methods described here.

For examples of how to defeat the use of restricted shells, one hackers guide to do this can be found at <https://www.metahackers.pro/breakout-of-restricted-shell/> and another at <https://www.exploit-db.com/docs/english/44592-linux-restricted-shell-bypass-guide.pdf>.

Use of a restricted shell may give a false sense of security.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Restricted Accounts

There are times when granting access to a user is necessary but is limited in scope. Setting up a restricted user account may be the solution.

- Use the restricted shell
- Limit available system programs and user applications
- Limit system resources
- Limit access times
- Limit access locations

When the restricted shell is invoked, it executes `$HOME/.bash.profile` without restriction. This is why the user must not have either write or execute permissions on the home directory.

Make sure that when you set up such an account that you do not inadvertently add system directories to the PATH environment variable, because this allows the restricted user the ability to execute other system programs, such as an unrestricted shell.

The restricted shell may be invoked with `/bin/bash -r`. Unfortunately, flags may not be specified in the `/etc/passwd` file. If you want to use a restricted shell, you need to set up a symbolic link or a hard linked file for the shell interpreter.

From the command line, or from a script, a restricted shell may be invoked with `/bin/bash -r`. However, flags may not be specified in the `/etc/passwd` file. A simple way to get around this restriction would be to do one of the following:

```
$ cd /bin ; sudo ln -s bash rbash
$ cd /bin ; sudo ln bash rbash
$ cd /bin ; sudo cp bash rbash
```

and then use `/bin/rbash` as the shell in `/etc/passwd`.

When setting up such an account, one should avoid inadvertently adding system directories to the PATH environment variable; this would grant the restricted user the ability to execute other system programs, such as an unrestricted shell.

Restricted accounts are also sometimes referred to as **limited accounts**.

## 30.9 The root Account

# The root Account

Use of the root account should be a rare exception

- Only log in as root when absolutely necessary
  - Avoid mistakes
  - Avoid potential security vulnerabilities
- Require use of **su** to acquire root privilege
  - Disallow direct root logins
  - Set up an audit trail of who became root
- For greater security use **sudo**

The root account should only be used for administrative purposes when absolutely necessary and never used as a regular account. Mistakes can be very costly, both for integrity and stability, and system security.

By default, root logins through the network are generally prohibited for security reasons. One can permit **Secure Shell** logins using **ssh**, which is configured with [`/etc/ssh/sshd\_config`](#), and **PAM** (Pluggable Authentication Modules), through the [`pam\_securetty.so`](#) module and the associated [`/etc/securetty`](#) file. Root login is permitted only from the devices listed in [`/etc/securetty`](#).

It is generally recommended that all root access be through **su**, or **sudo** (causing an audit trail of all root access through **sudo**). Note some distributions (such as **Ubuntu**), by default actually prohibit logging in directly to the root account.

**PAM** can also be used to restrict which users are allowed to **su** to root. It might also be worth it to configure **auditd** to log all commands executed as root.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 30.10 SSH

# SSH

- **ssh**: login to or carry out commands on remote system.
- Includes **scp**: transfer files to and from remote systems
- Uses strong encryption for security  
(hence name: **Secure SHell**)

```
$ ssh remote_computer.com
$ ssh some_user@remote_computer.com
$ ssh some_user@remote_computer.com apt-get update
$ scp file.txt remote_computer.com:/tmp
$ scp usrl@rem1.com:/tmp/f.txt usr2@rem2.com:/tmp/nf.txt
```

- Run a command on multiple systems simultaneously:

```
$ for machines in node1 node2 node3
 do
 (ssh $machines some_command &)
done
```

One often needs to login through the network into a remote system, either with the same user name or another. Or one needs to transfer files to and from a remote machine. In either case, one wants to do this securely, free from interception.

**SSH (Secure SHell)** exists for this purpose. It uses encryption based on strong algorithms. Assuming the proper **ssh** packages are installed on a system, one needs no further setup to begin using **ssh**. To sign onto a remote system:

```
$ ssh farflung.com
```

```
1 student@farflung.com's password: (type here)
```

assuming there is a student account on [farflung.com](#). To log in as a different user:

```
$ ssh root@farflung.com
$ ssh -l root farflung.com
```

To copy files from one system to another:

```
$ scp file.txt farflung.com:/tmp
$ scp file.tex student@farflung.com/home/student
$ scp -r some_dir farflung.com:/tmp/some_dir
```

(We have omitted the request for a password to save space.)

# ssh Configuration Files

- In home directory:
  - `id_rsa`: the user's **private** encryption key.
  - `id_rsa.pub`: the user's **public** encryption key.
  - `authorized_keys`: Public keys that are permitted to login.
  - `known_hosts`: Hosts from which logins have been allowed in the past.
  - `config`: File for specifying various options.
- Public and private keys generated with **ssh-keygen**

One can configure **SSH** further to expedite its use, in particular to permit logging in without a password. User-specific configuration files are created under every user's home directory in the hidden `.ssh` directory:

First a user has to generate their private and public encryption keys with **ssh-keygen**:

```
$ ssh-keygen
1 Generating public/private rsa key pair.
2 ...
```

The private key must **never** ever be shared with anyone. The public key, however, should be given to any machine with which you want to permit password-less access. It should also be added to your `authorized_keys` file, together with all the public keys from other users who have accounts on your machine and you want to permit password-less access to their accounts. `known_hosts` is gradually built up as **ssh** accesses occur. If the system detects changes in the users who are trying to log in through **ssh** it will warn you of them and afford the opportunity to deny access.

Note that `authorized_keys` contains information about users and machines:

```
$ cat authorized_keys
1 ssh-rsa AAAAB3Nza.....student@debian
2 ssh-rsa AAAAB3Nza.....student@fedora
```

while `known_hosts` contains only information about computer nodes:

```
$ cat known_hosts
1 x7 ecdsa-sha2-nistp256 AAAAE2VjZHNhLXN..... M=
2 192.168.1.200 ecdsa-sha2-nistp256 AAAA..... bcs=
```

Do `man ssh_config` to see the options that can go into the `ssh` configuration files.

**For the exclusive use of LFS301 corp class taught 06 to 09 December**

# Remote Graphical Login

- Login into remote machine with full graphical desktop
- Often use **VNC** (Virtual Network Computing)
- Common implementation is **tigervnc**
- As simple as:
  - On remote machine (e.g., `some_machine`):  
    \$ `vncserver`
  - On local machine:  
    \$ `vncviewer -via server student@some_machine localhost:2`

To test this first make sure you have the **vnc** packages installed:

```
$ which vncserver vncviewer
1 /usr/bin/vncserver
2 /usr/bin/vncviewer
```

If you do not find these programs you will have to install with something like:

```
$ sudo [dnf|zypper|apt-get] install tigervnc*
```

using the right package management system command. (The exact package name have varied between **Linux** distributions, so we are not giving exact package names. You may wind up installing more than you need, but the packages are not large.) Start the server as a normal user with:

```
$ vncserver
```

You can test with

```
$ vncviewer localhost:2
```

(You may have to play with numbers other than 2 (1, 3, 4, ...) depending on what you are running at the moment and how your machine is configured.) To view from a remote machine it is just slightly different:

```
$ vncviewer -via student@some_machine localhost:2
```

If you get a rather strange message about having to authenticate because of “color profile” and no passwords work, you have to kill the **colord** daemon on the server machine, as in

```
$ sudo systemctl stop colord
```

This is a bug (not a feature) and will only appear in some distributions and some systems for unclear reasons.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 30.11 Labs



### Video Demonstration Resources

`using_user_accounts_demo.mp4`

## Exercise 30.1: Working with User Accounts

1. Examine `/etc/passwd` and `/etc/shadow`, comparing the fields in each file, especially for the normal user account. What is the same and what is different?

2. Create a `user1` account using `useradd`.

3. Login as `user1` using `ssh`. You can just do this with:

```
$ ssh user1@localhost
```

It should fail because you need a password for `user1`; it was never established.

4. Set the password for `user1` to `user1pw` and then try to login again as `user1`.

5. Look at the new records which were created in the `/etc/passwd`, `/etc/group` and the `/etc/shadow` files.

6. Look at the `/etc/default/useradd` file and see what the current defaults are set to. Also look at the `/etc/login.defs` file.

7. Create a user account for `user2` which will use the **Korn** shell (**ksh**) as its default shell. (if you do not have `/bin/ksh` install it or use the **C** shell at `/bin/csh`.) Set the password to `user2pw`.

8. Look at `/etc/shadow`. What is the current expiration date for the `user1` account?

9. Use `chage` to set the account expiration date of `user1` to December 1, 2013.

Look at `/etc/shadow` to see what the new expiration date is.

10. Use `usermod` to lock the `user1` account.

Look at `/etc/shadow` and see what has changed about `user1`'s password. Reset the password to `user1pw` on the account to complete this exercise.

## ✓ Solution 30.1

1. `$ sudo grep student /etc/passwd /etc/shadow`

```
1 /etc/passwd:student:x:1000:100:LF Student:/home/student:/bin/bash
2 /etc/shadow:student:6jtoFVPICHhba$iGFFUU8ctr0GoistJ4/30DrNLi1FS66qnn0VbS6Mvm
3 luKI08SgbzT5.Ic0Ho5j/S0dCagZmF2RgzTvzLb11H0:16028:0:99999:7:::
```

(You can use any normal user name in the place of `student`.) About the only thing that matches is the user name field.

2. `$ sudo useradd user1`

3. `$ ssh user1@localhost`

```
1 user1@localhost's password:
```

Note you may have to first start up the `sshd` service as in:

```
$ sudo service sshd restart
```

or

For the exclusive use of LFS301 corp class taught 06 to 09 December

```
$ sudo systemctl restart sshd.service
4. $ sudo passwd user1
```

1 Changing password for user user1.  
2 New password:

```
5. $ sudo grep user1 /etc/passwd /etc/shadow
```

1 /etc/passwd:user1:x:1001:100::/home/user1:/bin/bash  
2 /etc/shadow:user1:\$6\$OBE1mPMw\$CIc7urbQ9ZSnyiniVOeJxKqLFu8fz4whfEexVem2  
3 TFpucuwRN1CCHZ19XGhj4qVujs1RIS.P4aCXd/y1U4utv.:16372:0:99999:7:::

6. On either **RHEL** or **openSUSE** systems for example:

```
$ cat /etc/default/useradd
1 # useradd defaults file
2 GROUP=100
3 HOME=/home
4 INACTIVE=-1
5 EXPIRE=
6 SHELL=/bin/bash
7 SKEL=/etc/skel
8 CREATE_MAIL_SPOOL=yes
```

```
$ cat /etc/login.defs
1
```

We don't reproduce the second file as it is rather longer, but examine it on your system.

```
7. $ sudo useradd -s /bin/ksh user2
$ sudo passwd user2
```

1 Changing password for user user2.  
2 New password:

```
8. $ sudo grep user1 /etc/shadow
```

1 user1:\$6\$OBE1mPMw\$CIc7urbQ9ZSnyiniVOeJxKqLFu8fz4whfEexVem2TFpucuwRN1CCHZ  
2 19XGhj4qVujs1RIS.P4aCXd/y1U4utv.:16372:0:99999:7:::

There should be no expiration date.

```
9. $ sudo chage -E 2013-12-1 user1
$ sudo grep user1 /etc/shadow
```

1 user1:\$6\$OBE1mPMw\$CIc7urbQ9ZSnyiniVOeJxKqLFu8fz4whfEexVem2TFpucuwRN1CCHZ  
2 19XGhj4qVujs1RIS.P4aCXd/y1U4utv.:16372:0:99999:7::16040:

```
10. $ sudo usermod -L user1
```

```
$ sudo passwd user1
```

## Exercise 30.2: Restricted Shells and Accounts

- Start a restricted shell in your current window with:

```
$ bash -r
```

Try elementary options such as resetting the path or changing directories.

- Set up a restricted account and verify its restricted nature, then clean up.

For the exclusive use of LFS301 corp class taught 06 to 09 December

**Please Note**

- On some distributions, notably some **Ubuntu**-based versions, there is a bug which prevents this lab from behaving properly. On **RedHat**-based distributions, the above correct behaviour is observed.
- As noted earlier, the use of restricted shells is deprecated as they are really not secure and there are better methods available. However, you may run into them which is why we discuss this facility.

**Solution 30.2**

1. c8:/tmp>bash -r  
c8:/tmp>cd \$HOME

1 | rbash: cd: restricted

c8:/tmp>PATH=\$PATH:/tmp

1 | rbash: PATH: readonly variable

c8/tmp>exit

1 | exit

2. c8/home/coop>sudo ln /bin/bash /bin/rbash  
c8:/home/coop>sudo useradd -s /bin/rbash fool  
c8:/home/coop>sudo passwd fool

1 | Changing password for user fool.  
2 | New password:  
3 | BAD PASSWORD: The password is shorter than 8 characters  
4 | Retype new password:  
5 | passwd: all authentication tokens updated successfully.

c8:/home/coop>sudo su - fool

1 | fool@c8~]\$ c d /tmp  
2 | -rbash: /usr/libexec/pk-command-not-found: restricted: cannot specify '/' in command names

[fool@c8~]\$ cd /tmp

1 | -rbash: cd: restricted

[fool@c8~]\$ PATH=\$PATH:/tmp

1 | -rbash: PATH: readonly variable

[fool@87 ~]\$ exit

1 | logout

c8/home/coop>sudo userdel -r fool  
c8/home/coop>sudo rm /bin/rbash

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 31

# Group Management



|      |                               |     |
|------|-------------------------------|-----|
| 31.1 | Groups . . . . .              | 386 |
| 31.2 | Group Management . . . . .    | 387 |
| 31.3 | User Private Groups . . . . . | 388 |
| 31.4 | Group Membership . . . . .    | 389 |
| 31.5 | Labs . . . . .                | 390 |

## 31.1 Groups

# Groups

- Allowing users to share a work area (directories, files etc.)
- Setting up file permissions to allow access to group members, but not the entire world
- Permitting certain specified users to access resources they would not be allowed to otherwise

**Linux** systems form collections of users called **groups**, whose members share some common purpose. To further that end, they share certain files and directories and maintain some common privileges; this separates them from others on the system, sometimes collectively called the **world**. Using groups aids collaborative projects enormously.

Users belong to one or more **groups**

Groups are defined in `/etc/group`, which has the same role for groups as `/etc/passwd` has for users. Each line of the file looks like:



`/etc/group`

```
....
groupname:password:GID:user1,user2,...
....
```

- `groupname` is the name of the group.
- `password` is the password place-holder. Group passwords may be set, but only if `/etc/gshadow` exists.
- `GID` is the group identifier. Values between 0 and 99 are for system groups. Values between 100 and `GID_MIN` (as defined in `/etc/login.defs` and usually the same as `UID_MIN`) are considered special. Values over `GID_MIN` are for **UPG** (User Private Groups).
- `user1,user2,...` is a comma-separated list of users who are members of the group. The user need not be listed here if this group is the user's principal group.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 31.2 Group Management

# Group Management

Group accounts may be managed and maintained with:

- **groupadd**: Add a new group
- **groupmod**: Modify a group and add new users
- **groupdel**: Remove a group
- **usermod**: Manage a user's group memberships

These group manipulation utilities modify `/etc/group` and (if it exists) `/etc/gshadow`, and may only be executed by root.

### Examples:

```
$ sudo groupadd -r -g 215 staff
$ sudo groupmod -g 101 blah
$ sudo groupdel newgroup
$ sudo usermod -G student,group1,group2 student
```

Note: Be very careful with the **usermod -G** command; the group list that follows is the complete list of groups, not just the changes. Any supplemental groups left out will be gone! Non-destructive use should utilize the **-a** option which will preserve pre-existing group memberships when adding new ones.

For the exclusive use of LFS301 corp class taught 06 to 09 December

### 31.3 User Private Groups

## User Private Groups

- **Linux** uses User Private Groups (UPG)
- By default users have the primary group id equal to their user id, and their group name the same as their username
- The **umask** is set to 002 for all users created with UPG (**umask** will be discussed in detail later.)
- User files are created with permissions 664 and directories 775

**Linux** uses **User Private Groups** (UPG).

The idea behind UPGs is that each user will have his or her own group. However, UPGs are **not guaranteed** to be private; additional members may be added to someone's private group in [`/etc/group`](#).

By default, users whose accounts are created with **useradd** have: primary GID = UID and the group name is also identical to the user name.

As specified in [`/etc/profile`](#), the **umask** is set to 002 for all users created with UPG. Under this scheme user files are thus created with permissions 664 (rw-rw-r----) and directories with 775 (rwxrwxr-x). (We will discuss **umask** later.)

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 31.4 Group Membership

# Group Membership

A **Linux** user:

- Has one primary group
  - Primary group is the one listed in `/etc/passwd`
  - Also will be listed in `/etc/group`
- May belong to between 0 and 15 secondary groups
- Can identify the groups he or she belongs to by executing either of

```
$ groups [user1 user2 ...]
$ id -Gn [user1 user2 ...]
```
- With no arguments, either command reports on the current user.

The primary group is the GID that is used whenever the user creates files or directories. Membership in other, secondary, groups grants the user those additional permissions.

Users can identify which groups they are a member of with either the **groups** or **id -Gn** commands.

Default groups can differ by distribution and installation specifics.

On **CentOS**:

```
[student@CentOS ~] groups
1 student
```

On **Ubuntu**:

```
student@ubuntu ~$ groups
1 student adm cdrom sudo dip plugdev lpadmin sambashare libvirt
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 31.5 Labs

### Exercise 31.1: Working with Groups

1. Create two new user accounts (`rocky` and `bullwinkle` in the below) and make sure they have home directories.
2. Create two new groups, `friends` and `bosses` (with a GID of 490). Look at `/etc/group`. See what GID was given to each new group.
3. Add `rocky` to both new groups.  
Add `bullwinkle` to group `friends`.  
Look in `/etc/group` to see how it changed.
4. Login as `rocky`. Create a directory called `somedir` and set the group ownership to `bosses`. (Using `chgrp` which will be discussed in the next session.)  
(You will probably need to add execute privileges for all on `rocky`'s home directory.)
5. Login as `bullwinkle` and try to create a file in `/home/rocky/somedir` called `somefile` using the `touch` command.  
Can you do this? No, because of the group ownership and the `chmod a+x` on the directory.
6. Add `bullwinkle` to the `bosses` group and try again. Note you will have to log out and log back in again for the new group membership to be effective. do the following:

### ✓ Solution 31.1

```
1. $ sudo useradd -m rocky
$ sudo useradd -m bullwinkle
$ sudo passwd rocky

1 Enter new UNIX password:
2 Retype new UNIX password:
3 passwd: password updated successfully
```

```
$ sudo passwd bullwinkle

1 Enter new UNIX password:
2 Retype new UNIX password:
3 passwd: password updated successfully
```

```
$ ls -l /home

1 total 12
2 drwxr-xr-x 2 bullwinkle bullwinkle 4096 Oct 30 09:39 bullwinkle
3 drwxr-xr-x 2 rocky rocky 4096 Oct 30 09:39 rocky
4 drwxr-xr-x 20 student student 4096 Oct 30 09:18 student
```

```
2. $ sudo groupadd friends
$ sudo groupadd -g 490 bosses
$ grep -e friends -e bosses /etc/group
```

```
1 friends:x:1003:
2 bosses:x:490:
```

```
3. $ sudo usermod -G friends,bosses rocky
$ sudo usermod -G friends bullwinkle
```

**For the exclusive use of LFS301 corp class taught 06 to 09 December**

```
1 rocky:x:1001:
2 bullwinkle:x:1002:
3 friends:x:1003:rocky,bullwinkle
4 bosses:x:490:rocky
```

```
$ groups rocky bullwinkle
```

```
1 rocky : rocky friends bosses
2 bullwinkle : bullwinkle friends
```

4. \$ ssh rocky@localhost

```
$ cd ~
$ mkdir somedir
$ chgrp bosses somedir
$ ls -l
```

```
1 total 16
2 -rw-r--r-- 1 rocky rocky 8980 Oct 4 2013 examples.desktop
3 drwxrwxr-x 2 rocky bosses 4096 Oct 30 09:53 somedir
```

```
$ chmod a+x .
```

5. \$ ssh bullwinkle@localhost

```
$ touch /home/rocky/somedir/somefile
```

```
1 touch: cannot touch /home/rocky/somedir/somefile: Permission denied
```

```
$ exit
```

6. \$ sudo usermod -a -G bosses bullwinkle

```
$ ssh bullwinkle@localhost
$ touch /home/rocky/somedir/somefile
$ ls -al /home/rocky/somedir
```

(note ownership of files)

---

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 32

# File Permissions and Ownership



|      |                                          |     |
|------|------------------------------------------|-----|
| 32.1 | File Permissions and Ownership . . . . . | 394 |
| 32.2 | File Access Rights . . . . .             | 395 |
| 32.3 | chmod, chown and chgrp . . . . .         | 396 |
| 32.4 | umask . . . . .                          | 399 |
| 32.5 | Filesystem ACLs . . . . .                | 400 |
| 32.6 | Labs . . . . .                           | 401 |

## 32.1 File Permissions and Ownership

# Owner, Group and World

- **owner**: the user who owns the file (also called **user**).
- **group**: the group of users who have access.
- **other**: the rest of the world (also called **world**).

```
$ ls -l
```

```
-rw-rw-r-- 1 student student 1601 Mar 9 15:04 mfile
-rw-r--r-- 1 student aproject 3280 Apr 15 5:09 gfile
-rwxr-xr-x 1 student bproject 803280 Dec 9 11.42 doit
```

When you do:

```
$ ls -l a_file
```

```
1 -rw-rw-r-- 1 coop aproject 1601 Mar 9 15:04 a_file
```

after the first character (which indicates the type of the file object) there are nine more which indicate the **access rights** granted to potential file users. These are arranged in three groups of three, corresponding to **owner**, **group** and **other** (same as **world**).

In the above listing, the user is **coop** and the group is **aproject**.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 32.2 File Access Rights

# File Access Rights

- If you do a long listing of a file, as in:

```
$ ls -l /usr/bin/vi
```

```
-rwxr-xr-x. 1 root root 910200 Jan 30 2014 /usr/bin/vi
```

Each of the triplets (in characters 2-10) can have each of the following set:

- **r**: read access is allowed.
- **w**: write access is allowed.
- **x**: execute access is allowed.
- Authentication is granted by the first to succeed of:
  - File ownership
  - Group membership
  - World permissions

If a permission is not allowed, a – appears instead of one of these characters.

In addition, other specialized permissions exist for each category, such as the **setuid/setgid** permissions.

These **file access permissions** are a critical part of the **Linux** security system. Any request to access a file requires comparison of the credentials and identity of the requesting user to those of the owner of the file.

This **authentication** is granted depending on one of these three sets of permissions, in the following order:

1. If the requester is the file owner, the file owner permissions are used.
2. Otherwise, if the requester is in the group that owns the files, the group permissions are examined;
3. If that does not succeed, the world permissions are examined.

For the exclusive use of LFS301 corp class taught 06 to 09 December

### 32.3 chmod, chown and chgrp

## chmod

- Changing file permissions is done with **chmod**.
- For example, to give the owner and world execute permission, and remove the group write permission:

```
$ ls -l a_file
-rw-rw-r-- 1 coop coop 1601 Mar 9 15:04 a_file

$ chmod uo+x,g-w a_file
$ ls -l a_file
-rwxr--r-x 1 coop coop 1601 Mar 9 15:04 a_file
```

where **u** stands for user (owner), **o** stands for other (world), and **g** stands for group.

- You can only change permissions on files you own, unless you are the superuser.
- Permissions are often given in numerical form, such as 644

Permissions can be represented either as a bitmap, usually written in octal, or in a symbolic form. Octal bitmaps usually look like 0755 while symbolic representations look like u+rwx,g+rwx,o+rwx.

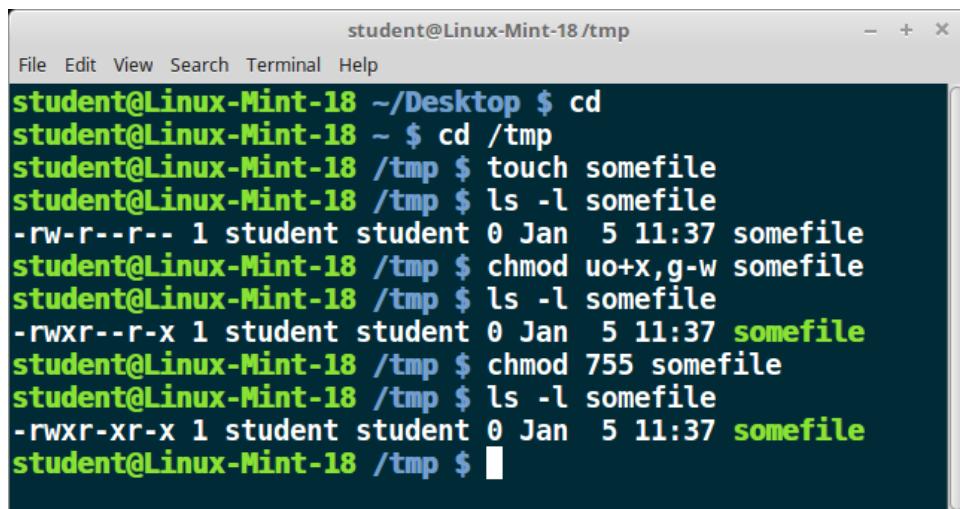
## Octal Digits

- The **Octal** number representation is the sum for each digit of:
  - 4 if read permission is desired.
  - 2 if write permission is desired.
  - 1 if execute permission is desired.

The symbolic syntax can be difficult to type and remember, so one often uses the octal shorthand which lets you set all the permissions in one step. This is done with a simple algorithm, and a single digit suffices to specify all three permission bits for each entity.

Thus 7 means read/write/execute, 6 means read/write, and 5 means read/execute.

When you apply this with **chmod** you have to give a value for each of the three digits, such as in the below example:



```
student@Linux-Mint-18 ~/Desktop $ cd
student@Linux-Mint-18 ~ $ cd /tmp
student@Linux-Mint-18 /tmp $ touch somefile
student@Linux-Mint-18 /tmp $ ls -l somefile
-rw-r--r-- 1 student student 0 Jan 5 11:37 somefile
student@Linux-Mint-18 /tmp $ chmod uo+x,g-w somefile
student@Linux-Mint-18 /tmp $ ls -l somefile
-rwxr--r-x 1 student student 0 Jan 5 11:37 somefile
student@Linux-Mint-18 /tmp $ chmod 755 somefile
student@Linux-Mint-18 /tmp $ ls -l somefile
-rwxr-xr-x 1 student student 0 Jan 5 11:37 somefile
student@Linux-Mint-18 /tmp $
```

Figure 32.1: Using chmod

For the exclusive use of LFS301 corp class taught 06 to 09 December

# chown and chgrp

- Change file ownership with **chown** (only superuser can do this)

```
$ sudo chown wally somefile
```

- Change group ownership with **chgrp** (only to groups you belong to)

```
$ sudo chgrp cleavers somefile
```

- Change both at the same time:

```
$ sudo chown wally:cleavers somefile
```

where you separate the owner and the group with a colon (or period).

- Can use the **-R** option for recursive:

```
$ sudo chown -R wally:cleavers ./
```

```
$ sudo chown -R wally:wally subdir
```

Changing file ownership is done with **chown** and changing the group is done with **chgrp**.

Only the superuser can change ownership on files. Likewise, you can only change group ownership to groups that you are a member of.

Use of the **-R** option ensures all files and directories underneath the argument have their ownership changed.

## 32.4 umask

# umask

- Used to set default permissions for files and directories
- **umask**: program to set the umask
  - Lists permissions to withhold
- Default umask set in `/etc/profile`
  - root's umask is 022
- Can be set by any user

The default permissions given when creating a file are read/write for owner, group and world (0666), and for a directory it is read/write/execute for everyone (0777). However, if you do the following:

```
$ touch afile
$ mkdir adir
$ ls -l | grep -e afile -e adir
1 drwxrwxr-x 2 coop coop 4096 Sep 16 11:18 adir
2 -rw-rw-r-- 1 coop coop 0 Sep 16 11:17 afile
```

you will notice the actual permissions have changed to 664 for the file and 775 for the directory. They have been modified by the current **umask**, whose purpose is to show which permissions should be denied. The current value can be shown by:

```
$ umask
```

```
1 0002
```

which is the most conventional value set by system administrators for users. This value is combined with the file creation permissions to get the actual result; i.e.,

`0666 & ~002 = 0664; i.e., rw-rw-r--`

You can change the **umask** at any time with the **umask** command, as in

```
$ umask 0022
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 32.5 Filesystem ACLs

# Filesystem ACLs

- **POSIX ACLs** (Access Control Lists) extend the simpler user, group, world system
- Used to share files/directories without using 777 permission
- Implemented as mount option (acl)
- Default set on filesystems created at install
- Use getfacl/setfacl to get/set **ACLs**
- Examples:

```
$ getfacl /home/stephane/file1
$ setfacl -m u:isabelle:rx /home/stephane/file1
$ setfacl -x u:isabelle /home/stephane/file
```

Particular privileges can be granted to specific users or groups of users when accessing certain objects or classes of objects. While the **Linux** kernel enables the use of **ACLs**, it still must be implemented as well in the particular filesystem. All major filesystems used in modern **Linux** distributions incorporate the **ACL** extensions, and one can use the option **-acl** when mounting. A default set of **ACLs** is created at system install.

Note that new files inherit the default **ACL** (if set) from the directory they reside in. Also note that **mv** and **cp -p** preserve **ACLs**.

To remove an **ACL**:

```
$ setfacl -x u:isabelle /home/stephane/file1
```

To set the default on a directory:

```
$ setfacl -m d:u:isabelle:rx somedir
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 32.6 Labs



### Video Demonstration Resources

[using\\_umask\\_demo.mp4](#)

#### Exercise 32.1: Using chmod

One can use either the octal digit or symbolic methods for specifying permissions when using **chmod**. Let's elaborate some more on the symbolic method.

It is possible to either give permissions directly, or add or subtract permissions.

Try the following examples:

```
$ chmod u=r,g=w,o=x afile
$ chmod u+w,g-w,+rw afile
$ chmod ug=rwx,o-rw afile
```

After each step do:

```
$ ls -l afile
```

to see how the permissions took, and try some variations.

#### Exercise 32.2: umask

Create an empty file with:

```
$ touch afile
$ ls -l afile
1 -rw-rw-r-- 1 coop coop 0 Jul 26 12:43 afile
```

which shows it is created by default with both read and write permissions for owner and group, but only read for world.

In fact, at the operating system level the default permissions given when creating a file or directory are actually read/write for owner, group **and** world (0666); the default values have actually been modified by the current **umask**.

If you just type **umask** you get the current value:

```
$ umask
```

```
1 0002
```

which is the most conventional value set by system administrators for users. This value is combined with the file creation permissions to get the actual result; i.e.,

0666 & ~002 = 0664; i.e., rw-rw-r--

Try modifying the **umask** and creating new files and see the resulting permissions, as in:

```
$ umask 0022
$ touch afile2
$ umask 0666
$ touch afile3
$ ls -l afile*
```

#### Exercise 32.3: Using Access Control Lists

For the exclusive use of LFS301 corp class taught 06 to 09 December

1. Create a file using your usual user name and run **getfacl** on it to see its properties.
2. Create a new user account with default properties (or reuse one from previous exercises).
3. Login as that user and try to add a line to the file you created in the first step. This should fail.
4. User **setfacl** to make the file writeable by the new user and try again.
5. User **setfacl** to make the file not readable by the new user and try again.
6. Clean up as necessary

## ✓ Solution 32.3

It is probably easiest to open two terminal windows, one to work in as your normal user account, and the other as the secondary one.

1. In window 1:

```
$ echo This is a file > /tmp/afile
$ getfacl /tmp/afile
1 getfacl: Removing leading '/' from absolute path names
2 # file: tmp/afile
3 # owner: coop
4 # group: coop
5 user::rw-
6 group::rw-
7 other::r--
```

2. In window 1:

```
$ sudo useradd fool
$ sudo passwd fool
...
```

3. In window 2:

```
$ sudo su - fool
$ echo another line > /tmp/afile
1 -bash: /tmp/afile: Permission denied
```

4. In window 1:

```
$ setfacl -m u:fool:rw /tmp/afile
$ getfacl /tmp/afile
1 getfacl: Removing leading '/' from absolute path names
2 # file: tmp/afile
3 # owner: coop
4 # group: coop
5 user::rw-
6 user:fool:rw-
7 group::rw-
8 mask::rwx
9 other::r--
```

In window 2:

```
$ echo another line > /tmp/afile
```

5. In window 1:

**For the exclusive use of LFS301 corp class taught 06 to 09 December**

```
$ setfacl -m u:fool:w /tmp/afile
```

In window 2:

```
$ echo another line > /tmp/afile
-bash: /tmp/afile: Permission denied
```

6. Cleaning up:

```
$ rm /tmp/afile
$ sudo userdel -r fool
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 33

# Pluggable Authentication Modules (PAM)



|      |                                                  |     |
|------|--------------------------------------------------|-----|
| 33.1 | PAM (Pluggable Authentication Modules) . . . . . | 406 |
| 33.2 | Authentication Process . . . . .                 | 407 |
| 33.3 | Configuring PAM . . . . .                        | 408 |
| 33.4 | LDAP Authentication ** . . . . .                 | 409 |



### Please Note

\*\* These sections may be considered in part or in whole as optional. They contain either background reference material, specialized topics, or advanced subjects. The instructor may choose to cover or not cover them depending on classroom experience and time constraints.

### 33.1 PAM (Pluggable Authentication Modules)

## Pluggable Authentication Modules (PAM)

- PAM controls authentication of users
- Has the following components
  - **PAM**-aware applications.
  - Configuration files in `/etc/pam.d/`.
  - **PAM** modules in the `libpam*` libraries, found in different locations depending on distribution.

Historically, authentication of users was performed individually by individual applications; i.e., `su`, `login`, and `ssh` would each authenticate and establish user accounts independently of each other.

Most modern **Linux** applications have been written or rewritten to exploit **PAM** (Pluggable Authentication **Modules**) so that authentication can be done in one uniform way, using `libpam`.

This library of modules provides enormous flexibility and consistency with respect to authentication, password, session, and account services.

Each **PAM**-aware application, or service, may be configured with respect to **PAM** by an individual configuration file in `/etc/pam.d/`.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 33.2 Authentication Process

# Authentication and Configuration Files

- A user invokes a **PAM**-aware application, such as **login**, **ssh**, or **su**.
- The application calls **libpam**.
- The library checks for files in `/etc/pam.d`; these delineate which **PAM** modules to invoke, including **system-auth**.
- Each referenced module is executed in accordance with the rules of the relevant configuration file for that application.

Each file in `/etc/pam.d` corresponds to a **service** and each (non-commented) line in the file specifies a rule. The rule is formatted as a list of space separated tokens, the first two of which are case insensitive:

```
type control module-path module-arguments
```

As an example, here are the contents of `/etc/pam.d/su` on an **RHEL** system:

```
File Edit View Search Terminal Help
c7:/tmp>cat /etc/pam.d/su
#%PAM-1.0
auth sufficient pam_rootok.so
Uncomment the following line to implicitly trust users in the "wheel" group.
#auth sufficient pam_wheel.so trust use_uid
Uncomment the following line to require a user to be in the "wheel" group.
#auth required pam_wheel.so use_uid
auth substack system-auth
auth include postlogin
account sufficient pam_succeed_if.so uid = 0 use_uid quiet
account include system-auth
password include system-auth
session include system-auth
session include postlogin
session optional pam_xauth.so
c7:/tmp>
```

Figure 33.1: PAM Configuration Files

Notice there is a stack here; **su** will require loading of **system-auth** etc.

For the exclusive use of LFS301 corp class taught 06 to 09 December

### 33.3 Configuring PAM

## PAM Rules

- `module-type control-flag pam-module`
- `module-type` is one of:
  - auth
  - account
  - password
  - session
- `control-flag` is one of:
  - required
  - requisite
  - optional
  - sufficient
- `pam-module`: any module name with parameters

The module type controls the step of the authentication process.

`auth`: Instructs the application to prompt the user for identification (username, password, etc). May set credentials and grant privileges.

`account`: Checks on aspects of the user's account such as password aging, access control.

`password`: Responsible for updating the user authentication token, usually a password.

`session`: Used to provide functions before and after the session is established (such as setting up environment, logging, etc.).

The control flag controls how the success or failure of a module affects the overall authentication process.

`required`: Must return success for the service to be granted. If part of a stack, all other modules are still executed. Application is not told which module or modules failed.

`requisite`: Same as required, except a failure in any module terminates the stack and a return status is sent to the application.

`optional`: Module is not required. If it is the only module then its return status to application may cause failure.

`sufficient`: If this module succeeds then no subsequent modules in the stack are executed. If it fails then it does not necessarily cause the stack to fail unless it is the only one in the stack.

`Pam-module` is a module in [/lib\[64\]/security](#)

There are other control flags such as `include` and `substack`.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 33.4 LDAP Authentication \*\*

# LDAP Authentication

- Centralized authentication
- Uses PAM
- Use **system-config-authentication** or **authconfig-tui**
  - Enable LDAP
  - Specify server, search base DN, and TLS
- Requires **openldap-clients**, **pam\_ldap**, **nss-pam-ldapd**

The **LDAP** (Lightweight Directory Access Protocol) is an industry standard protocol for using and administering distributed directory services over the network, and is meant to be both open and vendor-neutral.

When using **LDAP** for centralized authentication, each system (or client) connects to a centralized **LDAP** server for user authentication. Using **TLS** (Transport Layer Security) makes it a secure option and is recommended.

**LDAP** uses **PAM** and **system-config-authentication** or **authconfig-tui**. One has to specify the server, search base **DN** (domain name) and **TLS**. Also required is **openldap-clients**, **pam\_ldap** and **nss-pam-ldapd**.

When you configure a system for **LDAP** authentication, five files are changed:

```
/etc/openldap/ldap.conf
/etc/pam_ldap.conf
/etc/nslcd.conf
/etc/sssd/sssd.conf
/etc/nsswitch.conf
```

You can edit these files manually or use one of the utility programs available (**system-config-authentication** or **authconfig-tui**).

There is no lab to complete for this chapter.

For the exclusive use of LFS301 corp class taught 06 to 09 December

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 34

# Network Addresses



|      |                              |     |
|------|------------------------------|-----|
| 34.1 | IP Addresses . . . . .       | 412 |
| 34.2 | IPv4 Address Types . . . . . | 413 |
| 34.3 | IPv6 Address Types . . . . . | 415 |
| 34.4 | IP Address Classes . . . . . | 416 |
| 34.5 | Netmasks . . . . .           | 417 |
| 34.6 | Hostnames . . . . .          | 418 |
| 34.7 | Labs . . . . .               | 419 |

## 34.1 IP Addresses

# IP Addresses

- IP addresses are used to uniquely identify nodes across the internet.
- IP addresses are registered through **ISPs**
- An **IPv4** 32 bit address is composed of **4 octets**  
`148.114.252.10`  
(An octet is just a byte, 8 bits.)
- An **IPv6** 128 bit address is composed of **8 16-bit octet pairs**  
`2003:0db5:6123:0000:1f4f:0000:5529:fe23`
- There is a set of reserved addresses
- We will focus on **IPv4**, still most common in use

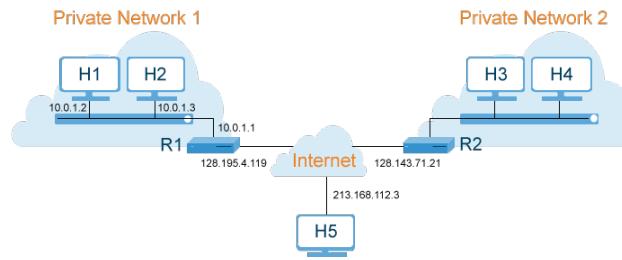


Figure 34.1: IP Addresses

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 34.2 IPv4 Address Types

# IPv4 Address Types

- Unicast
- Network
- Broadcast
- Multicast

IPv4 address types include:

- **Unicast:**

An address associated with a specific host. It might be something like 140.211.169.4 or 64.254.248.193.

- **Network:**

An address whose **host** portion is set to all binary zeroes. Ex. 192.168.1.0. (The host portion can be the last 1-3 octets as discussed later; here it is just the last octet.)

- **Broadcast:**

An address to which each member of a particular network will listen. It will have the host portion set to all 1 bits, such as in 172.16.255.255 or 148.114.255.255 or 192.168.1.255. (The host portion is the last two octets in the first two cases, just the last one in the third case.)

- **Multicast:**

An address to which appropriately configured nodes will listen. The address 224.0.0.2 is an example of a multicast address. Only nodes specifically configured to pay attention to a specific multicast address will interpret packets for that multicast group.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Reserved Addresses

- Loopback interfaces:  
127.x.x.x
- Address not given yet:  
0.0.0.0
- Generic Broadcast Private Address  
255.255.255.255
- Others  
10.0.0.0 – 10.255.255.255  
172.16.0.0 – 172.31.255.255  
192.168.0.0 – 192.168.255.255

Certain addresses and address ranges are reserved for special purposes:

- 127.x.x.x  
Reserved for the loopback (local system) interface, where  $0 \leq x \leq 254$ . Generally, 127.0.0.1.
- 0.0.0.0  
Used by systems that do not yet know their own address. Protocols like **DHCP** and **BOOTP** use this address when attempting to communicate with a server.

- 255.255.255.255  
Generic broadcast private address, reserved for internal use. These addresses are never assigned or registered to anyone. They are generally not routable.
- Other examples of reserved address ranges include:  
10.0.0.0 – 10.255.255.255  
172.16.0.0 – 172.31.255.255  
192.168.0.0 – 192.168.255.255

Each of these has a purpose. For example, the familiar address range, 192.168.x.x is used only for local communications within a private network.

You can see a long list of reserved addresses for both **IPv4** and **IPv6** at [https://en.wikipedia.org/wiki/Reserved\\_IP\\_addresses](https://en.wikipedia.org/wiki/Reserved_IP_addresses).

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 34.3 IPv6 Address Types

# IPv6 Address Types

- Unicast
- Multicast
- Anycast
- IPv4-mapped

**IPv6** address types include:

- **Unicast:**

A packet is delivered to one interface.

- **Link-local:** Autoconfigured for every interface to have one. Non-routable.
- **Global:** Dynamically or manually assigned. Routable.
- Reserved for documentation.

- **Multicast:**

A packet is delivered to multiple interfaces.

- **Anycast:**

A packet is delivered to the nearest of multiple interfaces (in terms of routing distance).

- **IPv4-mapped:**

An **IPv4** address mapped to **IPv6**. For example, `::FFFF:a.b.c.d/96`

In addition, **IPv6** has some special types of addresses such as loopback, which is assigned to the `lo` interface, as `::1/128`

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 34.4 IP Address Classes

# IPv4 Address Classes

- Class A – Highest order octet range 1-127
  - 128 networks, 16,772,214 hosts per network, 127.x.x.x reserved for loopback
- Class B – Highest order octet range 128-191
  - 16,384 networks, 65,534 hosts per network
- Class C – Highest order octet range 192-223
  - 2,097,152 networks, 254 hosts per network
- Class D – Highest order octet range 224-239
  - Multicast addresses
- Class E – Highest order octet range 240-254
  - reserved address range

Historically, IP addresses are based on defined classes. Classes A, B, and C are used to distinguish a network portion of the address from a host portion of the address. This is used for routing purposes.

The Class A addresses use 8 bits for the network portion of the address and 24 bits for the host portion of the address. The Class B addresses use 16 and 16 bits respectively while the Class C addresses use 24 bits for the network portion and 8 bits for the host portion.

Class D addresses are used for multicasting. Class E addresses are currently not used.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 34.5 Netmasks

# Netmasks

- Class A netmask
  - 255.0.0.0 in decimal
  - ff:00:00:00 in hex
  - 11111111 00000000 00000000 00000000 in binary
- Class B netmask
  - 255.255.0.0 in decimal
  - ff:ff:00:00 in hex
  - 11111111 11111111 00000000 00000000 in binary
- Class C netmask
  - 255.255.255.0 in decimal
  - ff:ff:ff:00 in hex
  - 11111111 11111111 11111111 00000000 in binary

The **netmask** is used to determine how much of the address is used for the network portion and how much for the host portion as we have seen. It is also used to determine network and broadcast addresses.

- Class A addresses use 8 bits for the network portion of the address and 24 bits for the host portion of the address.
- Class B addresses use 16 for the network and 16 for the host.
- Class C addresses use 24 for the network and 8 for the host.
- Class D addresses are used for multicasting.
- Class E addresses are currently not used.

We can calculate the network address by anding (logical and - &) the ip address and the netmask. We are interested in the network address because they define a local network which consists of a collection of nodes connected via the same media and sharing the same network address. All nodes on the same network can directly see each other.

```
ex. 172.16.2.17 ip address
&255.255.0.0 netmask

172.16.0.0 network address
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 34.6 Hostnames

# Getting and Setting the Hostname

- Label to distinguish a networked device from other nodes
- Historically also called **nodename**
- Set at install, but can always be changed later
- Any user can get the hostname with:

```
$ hostname
```

wally

- Changing hostname requires root privilege:

```
$ sudo hostname lumpy
```

lumpy

- The current value is always stored in [/etc/hostname](#) on most **Linux** distributions.
- To do this **persistently** so changes survive a reboot use **hostnamectl**, part of the **systemd** infrastructure

```
$ sudo hostnamectl set-hostname lumpy
```

For **DNS** purposes, hostnames are appended with a period (dot) and a domain name, so that a machine with a hostname of `antje` could have a **fully qualified domain name (FQDN)** of `antje.linuxfoundation.org`.

Historically, making persistent changes involved changing configuration files in the `etc` directory tree. On **Red Hat**-based systems this was `/etc/sysconfig/network`, on **Debian**-based systems this was `/etc/hostname` and on **SUSE**-based systems it was `/etc/HOSTNAME`. However one should use **hostnamectl** on modern systems:

```
1 $ hostnamectl
2 Static hostname: c8
3 Icon name: computer-desktop
4 Chassis: desktop
5 Machine ID: ce0c82382a8a4c80bb6931a917a2f1c
6 Boot ID: 94207b3fbdb9b4891b9a94e21762a47cb
7 Operating System: Red Hat Enterprise Linux 8.2 (Ootpa) \
8 dracut-049-70.git20200228.e18 (Initramfs)
9 Kernel: Linux 5.11.6
10 Architecture: x86-64
```

and to see a usage message:

```
1 $ hostnamectl --help
2
3 hostnamectl [OPTIONS...] COMMAND ...
4
5 Query or change system hostname.
```

|    |                                     |                                     |
|----|-------------------------------------|-------------------------------------|
| 6  | <code>-h --help</code>              | Show this help                      |
| 7  | <code>--version</code>              | Show package version                |
| 8  | <code>--no-ask-password</code>      | Do not prompt for password          |
| 9  | <code>-H --host=[USER@]HOST</code>  | Operate on remote host              |
| 10 | <code>-M --machine=CONTAINER</code> | Operate on local container          |
| 11 | <code>--transient</code>            | Only set transient hostname         |
| 12 | <code>--static</code>               | Only set static hostname            |
| 13 | <code>--pretty</code>               | Only set pretty hostname            |
| 14 | <br>Commands:                       |                                     |
| 15 | <code>status</code>                 | Show current hostname settings      |
| 16 | <code>set-hostname NAME</code>      | Set system hostname                 |
| 17 | <code>set-icon-name NAME</code>     | Set icon name for host              |
| 18 | <code>set-chassis NAME</code>       | Set chassis type for host           |
| 19 | <code>set-deployment NAME</code>    | Set deployment environment for host |
| 20 | <code>set-location NAME</code>      | Set location for host               |
| 21 | <code>set-location NAME</code>      | Set location for host               |

## 34.7 Labs



### Please Note

There are no lab exercises in this chapter. It just sets the stage for the following section on network configuration, which has several labs.

For the exclusive use of LFS301 corp class taught 06 to 09 December

---

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 35

# Network Devices and Configuration



|       |                                                      |     |
|-------|------------------------------------------------------|-----|
| 35.1  | Network Devices . . . . .                            | 422 |
| 35.2  | ip . . . . .                                         | 423 |
| 35.3  | ifconfig . . . . .                                   | 425 |
| 35.4  | Predictable Network Interface Device Names . . . . . | 427 |
| 35.5  | Network Configuration Files . . . . .                | 428 |
| 35.6  | Network Manager . . . . .                            | 429 |
| 35.7  | Routing . . . . .                                    | 434 |
| 35.8  | DNS and Name Resolution . . . . .                    | 437 |
| 35.9  | Network Diagnostics . . . . .                        | 440 |
| 35.10 | Labs . . . . .                                       | 444 |

## 35.1 Network Devices

# Network Devices

- Names usually consist of a type identifier followed by a number:
  - **Ethernet**: eth0, eth1, eno1, eno2, etc.
  - **Wireless**: wlan0, wlan1, wlp3s0, wlp3s2, etc
  - **Bridged**: br0, br1, etc.
  - **Virtual**: vmnet1, vmnet8, etc.
- One device can have multiple IP addresses
  - Older systems often had associations like: eth0:4 → eth0
  - Newer systems using **ip** instead of **ifconfig** just assign multiple addresses to one interface name

Unlike block and character devices, network devices are not associated with **special device files**, also known as **device nodes**. Rather than having associated entries in the `/dev` directory, they are known by their names.

Historically, multiple virtual devices could be associated with single physical devices; these were named with colons and numbers, so `eth0:0` would be the first alias on the `eth0` device. This was done to support multiple IP addresses on one network card. However, with the use of **ip** instead of **ifconfig**, this method is deprecated and we will not pursue it. this is not compatible with **IPV6**

The historical device naming conventions encounter difficulties, particularly when multiple interfaces of the same type were present. For example, suppose one has two network cards; one would be named `eth0` and the other `eth1`. However, which physical device should be associated with each name?

The simplest method would be to have the first device found be `eth0`, the second `eth1` etc. Unfortunately, probing for devices is not deterministic for modern systems, and devices may be located or plugged in an unpredictable order. Thus, one might wind up with the Internet interface swapped with the local interface. Even if hardware does not change, the order in which interfaces are located has been known to vary with kernel version and configuration.

Many system administrators have solved this problem in a simple manner, by hard-coding associations between hardware (MAC) addresses and device names in system configuration files and startup scripts. While this method has worked for years it requires manual tuning and had other problems, such as when MAC addresses were not fixed; this can happen in both embedded and virtualized systems.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 35.2 ip

# ip

- Used to configure, control, and query interface parameters and manipulate:
    - Devices
    - Routing
    - Tunnels
  - Basic syntax:
- ```
ip [ OPTIONS ] OBJECT { COMMAND | help }
ip [ -force ] -batch filename
```
- where the second form can read commands from a designated file.
- Works with **netlink** sockets, not **ioctl** system calls
 - Newer and more versatile than **ifconfig**

ip is the preferred command line utility as compared to the venerable **ifconfig** we discuss next. It is more versatile as well as more efficient because it uses **netlink** sockets rather than **ioctl** system calls.

ip can be used for a wide variety of tasks. It can be used to display and control devices, routing, policy-based routing, and tunneling.

The basic syntax is:

```
ip [ OPTIONS ] OBJECT { COMMAND | help }
ip [ -force ] -batch filename
```

where the second form can read commands from a designated file.

ip is a multiplex utility; the **OBJECT** argument describes what kind of action is going to be performed. The possible **COMMANDS** depend on which **OBJECT** is selected. Here are some of the main values of **OBJECT**:

Table 35.1: ip

OBJECT	Function
address	IPv4 or IPv6 protocol device address
link	Network Devices
maddress	Multicast Address
monitor	Watch for netlink messages
route	Routing table entry
rule	Rule in the routing policy database
tunnel	Tunnel over IP

For the exclusive use of LFS301 corp class taught 06 to 09 December

Using ip: Examples

- Show information for all interfaces:

```
$ ip link show
```

- Show information for eth0 interface, including statistics:

```
$ ip -s link show eth0
```

- Set ip address for eth0:

```
$ sudo ip addr add 192.168.1.7 dev eth0
```

- Bring interface eth0 down:

```
$ sudo ip link set eth0 down
```

- Set MTU to 1480 for interface eth0

```
$ sudo ip link set eth0 mtu 1480
```

- Set route to network:

```
$ sudo ip route add 172.16.1.0/24 via 192.168.1.5
```

```
student@ubuntu:~$ ip -s link show ens33
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
    link/ether 00:0c:29:c8:bf:ca brd ff:ff:ff:ff:ff:ff
      RX: bytes packets errors dropped overrun mcast
        1582717   1166     0     0     0
      TX: bytes packets errors dropped carrier collsns
        59159     746     0     0     0
student@ubuntu:~$ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
      inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
      inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:c8:bf:ca brd ff:ff:ff:ff:ff:ff
      inet 172.16.2.128/24 brd 172.16.2.255 scope global dynamic ens33
        valid_lft 1096sec preferred_lft 1096sec
      inet6 fe80::bf00:6f80:9502:f589/64 scope link
        valid_lft forever preferred_lft forever
student@ubuntu:~$
```

Figure 35.1: Using ip

For the exclusive use of LFS301 corp class taught 06 to 09 December

35.3 ifconfig

ifconfig

- Used to configure, control, and query interface parameters
- Being superseded by **ip**:
 - Some newer **Linux** distributions do not install by default
- Used from the command line


```
$ ifconfig
$ ifconfig eth0
$ ifconfig eth0 192.168.1.50
$ ifconfig eth0 netmask 255.255.255.0
$ ifconfig eth0 down
$ ifconfig eth0 mtu 1480
```
- Also called by network startup scripts

```
File Edit View Search Terminal Help
x7:/tmp>/sbin/ifconfig
enp0s31f6: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
          inet 192.168.1.250  netmask 255.255.255.0  broadcast 192.168.1.255
              ether 54:ee:75:b5:f8:39  txqueuelen 1000  (Ethernet)
              RX packets 251841  bytes 305672016 (291.5 MiB)
              RX errors 0  dropped 0  overruns 0  frame 0
              TX packets 148108  bytes 105257478 (100.3 MiB)
              TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0
              device interrupt 16  memory 0xf1200000-f1220000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
      inet 127.0.0.1  netmask 255.0.0.0
          loop  txqueuelen 1000  (Local Loopback)
          RX packets 100  bytes 7820 (7.6 KiB)
          RX errors 0  dropped 0  overruns 0  frame 0
          TX packets 100  bytes 7820 (7.6 KiB)
          TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

wlp4s0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
          ether 7e:6d:db:1e:16:9c  txqueuelen 1000  (Ethernet)
          RX packets 140946  bytes 192702080 (183.7 MiB)
          RX errors 0  dropped 0  overruns 0  frame 0
          TX packets 44664  bytes 7435027 (7.0 MiB)
          TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

x7:/tmp>
```

Figure 35.2: ifconfig Output

For the exclusive use of LFS301 corp class taught 06 to 09 December

ifconfig Examples

- Display information about all interfaces, or just one:

```
$ ifconfig  
$ ifconfig eth0
```

- Set the IP address to 192.168.1.50 on interface eth0:

```
$ sudo ifconfig eth0 192.168.1.50
```

- Set the netmask to 24-bit:

```
$ sudo ifconfig eth0 netmask 255.255.255.0
```

- Bring interface eth0 up or down:

```
$ sudo ifconfig eth0 up  
$ sudo ifconfig eth0 down
```

- Set the **MTU** (**M**aximum **T**ransfer **U**nit) to 1480 bytes for interface eth0:

```
$ sudo ifconfig eth0 mtu 1480
```

ifconfig is a system administration utility long found in **UNIX**-like operating systems to configure, control, and query network interface parameters from either the command line or from system configuration scripts.

It is also often called by network startup scripts.

The above slide provides some usage examples.

35.4 Predictable Network Interface Device Names

Predictable Network Interface Device Names

- PNIDN (Predictable Network Interface Device Names) is connected to **udev** and integration with **systemd**.
- There are now 5 types of names that devices can be given:
 1. Incorporating **Firmware** or **BIOS** provided index numbers for on-board devices. Example: eno1
 2. Incorporating **Firmware** or **BIOS** provided **PCI Express** hotplug slot index numbers. Example: ens1
 3. Incorporating physical and/or geographical location of the hardware connection. Example: enp2s0
 4. Incorporating the MAC address. Example: enx7837d1ea46da
 5. Using the old classic method. Example: eth0

For example, on a machine with two onboard **PCI** network interfaces that would have been eth0 and eth1:

```
$ ip link show | grep enp
2: enp4s2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN mode DEFAULT qlen 1000
3: enp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT qlen 1000

$ ifconfig | grep enp
enp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
enp4s2: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
```

These names are correlated with the physical locations of the hardware on the **PCI** system:

```
$ lspci | grep Ethernet
02:00.0 Ethernet controller: Marvell Technology Group Ltd. 88E8056 PCI-E Gigabit Ethernet Controller (rev 12)
04:02.0 Ethernet controller: Marvell Technology Group Ltd. 88E8001 Gigabit Ethernet Controller (rev 14)
```

The triplet of numbers at the beginning of each line from the **lspci** output is the bus, device (or slot), and function of the device; hence it reveals the physical location.

Likewise, for a wireless device that previously would have been simply named wlan0:

```
$ ip link show | grep wl
3: wlp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DORMANT qlen 1000

$ lspci | grep Centrino
03:00.0 Network controller: Intel Corporation Centrino Advanced-N 6205 [Taylor Peak] (rev 34)
```

We see the same pattern. It is easy to turn off the new scheme and go back to the classic names. We will leave that as a research project. In what follows we will mostly follow the classic names for definiteness and simplicity.

= For the exclusive use of LFS301 corp class taught 06 to 09 December

35.5 Network Configuration Files

Network Configuration Files

- Each distribution has its own set of files and/or directories:
 - **Red Hat**
`/etc/sysconfig/network`
`/etc/sysconfig/network-scripts/ifcfg-ethX`
`/etc/sysconfig/network-scripts/ifcfg-ethX:Y`
`/etc/sysconfig/network-scripts/route-ethX`
 - **Debian**
`/etc/network/interfaces`
 - **SUSE**
`/etc/sysconfig/network`
- With **systemd** getting more standardized
- Often preferable to use **Network Manager**
- On newer distributions these files may be gone or empty

On newer **Linux** distributions these configuration files are either non-existent, empty, or much smaller.

For the exclusive use of LFS301 corp class taught 06 to 09 December

35.6 Network Manager

Network Manager

- Network interfaces used to be mostly static, both in hardware and software
- Modern systems often have dynamic configurations:
 - Networks may change as a device is moved from place to place
 - Wireless devices may have a large choice of networks to hook into
 - Devices may change as hardware such as wireless devices, are plugged in or turned on and off
- Previous discussed configuration files were created to deal with more static situations and are very distribution dependent.
- **Network Manager** should be almost the same on different systems

Once upon a time, network connections were almost all wired (Ethernet) and did not change unless there was a significant change to the system.

As a system was booted, it consulted the network configuration files in the `/etc` directory subtree in order to establish the interface properties such as static or dynamic (DCHP) address configuration, whether the device should be started at boot etc.

If there were multiple network devices, policies had to be established as to what order they would be brought up, which networks they would connect to, what they would be called etc.

As wireless connections became more common (as well as hotplug network devices such as on **USB** adapters), configuration became much more complicated, both because of the transient nature of the hardware and that of the specific networks being connected to.

A step away from distribution dependent interfaces and configuration files was a big advance.

While **Network Manager** still uses configuration files, it is usually best to rely on its various utilities for manipulating and updating them.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Network Manager Interfaces

- **GUI**

Every **Linux** desktop (**GNOME**, **KDE**, **XFCE**, etc) has one, usually reached by clicking on something in the horizontal panel.

- **nmtui**

network manager text user interface is a rather simple navigable command line interface using the **ncurses** library interface

- **nmcli**

network manager command line interface is a lower level method which the higher level interfaces can use. It is useful for scripts

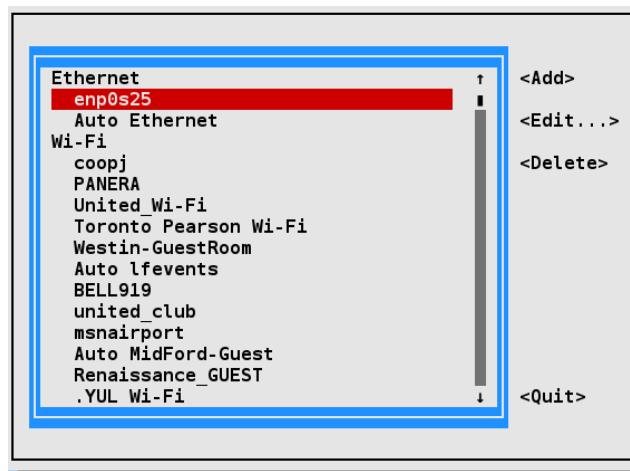
If you are using your laptop in a hotel room or a coffee shop you are probably going to use whatever graphical interface your **Linux** distribution's desktop offers. You can use this to select between different networks, configure security and passwords, turn devices off and on etc.

If you are making a configuration change on your system that is likely to last for a while you are likely to use **nmtui** as it has almost no learning curve and will edit the underlying configuration files for you.

If you need to run scripts that change the network configuration, you will want to use **nmcli**. Or, if you are a command line junkie, you may want to use this instead of **nmtui**.

If the **GUI** is properly done you should be able to accomplish any task using any of these three methods. However, we will focus on **nmtui** and **nmcli** because they are distribution independent and hide any differences in underlying configuration files.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Figure 35.3: **nmtui Main Screen**Figure 35.4: **nmtui Edit Screen**

nmtui is rather straightforward to use. You can navigate with either the arrow keys or the tab key.

Besides activating or editing connections, you also set the system hostname. However, some operations, such as this, can not be done by normal users and you will be prompted for the root password to go forward.

nmtui Wireless Configuration

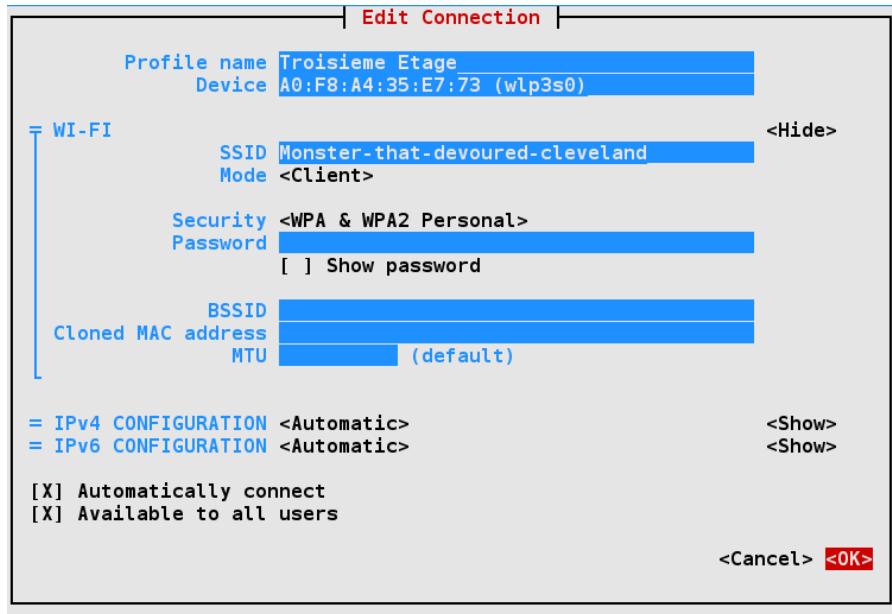


Figure 35.5: nmtui Wireless Configuration

For the exclusive use of LFS301 corp class taught 06 to 09 December

nmcli

- Command line interface to **Network Manager**
- Also has interactive mode
- See <https://fedoraproject.org/wiki/Networking/CLI>
- Also `man nmcli-examples`
- Will try some examples in an exercise

```
student@ubuntu:~$ nmcli --help
Usage: nmcli [OPTIONS] OBJECT { COMMAND | help }

OPTIONS
  -t[erse]                                terse output
  -p[retty]                                 pretty output
  -m[mode] tabular|multiline                output mode
  -c[olors] auto|yes|no                     whether to use colors in output
  -f[ields] <field1,field2,...>|all|common  specify fields to output
  -e[scape] yes|no                          escape columns separators in values
  -a[sk]                                     ask for missing parameters
  -s[how-secrets]                           allow displaying passwords
  -w[ait] <seconds>                         set timeout waiting for finishin
g operations
  -v[ersion]                                show program version
  -h[elp]                                    print this help

OBJECT
  g[eneral]      NetworkManager's general status and operations
  n[etworking]   overall networking control
  r[adio]        NetworkManager radio switches
  c[onnection]   NetworkManager's connections
  d[evice]       devices managed by NetworkManager
  a[gent]         NetworkManager secret agent or polkit agent
  m[onitor]     monitor NetworkManager changes

student@ubuntu:~$
```

Figure 35.6: **nmcli**

For the exclusive use of LFS301 corp class taught 06 to 09 December

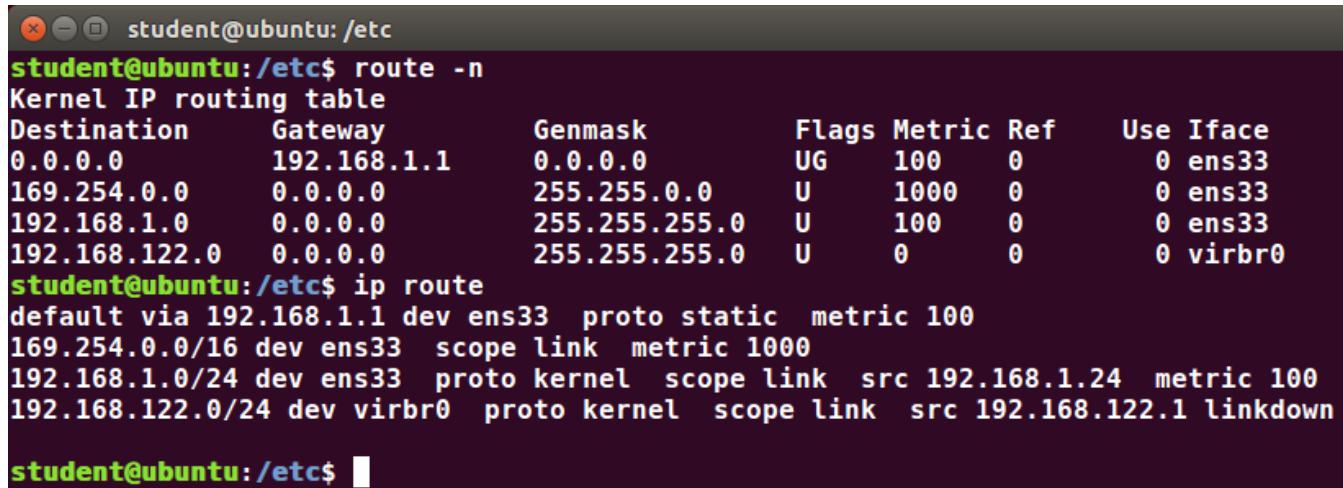
35.7 Routing

Routing

- Moving packets about and between networks
- Routing table
 - Defines paths to all networks and hosts
 - Local packets sent directly
 - Remote traffic sent to routers

Routing is the process of selecting paths in a network along which to send network traffic. The **routing table** is a list of routes to other networks managed by the system. It defines paths to all networks and hosts, sending remote traffic to routers.

To see the current routing table, one can use **route** or **ip**:



```
student@ubuntu:/etc$ route -n
Kernel IP routing table
Destination      Gateway        Genmask        Flags Metric Ref    Use Iface
0.0.0.0          192.168.1.1   0.0.0.0        UG    100    0        0 ens33
169.254.0.0      0.0.0.0       255.255.0.0   U     1000   0        0 ens33
192.168.1.0      0.0.0.0       255.255.255.0  U     100    0        0 ens33
192.168.122.0    0.0.0.0       255.255.255.0  U     0      0        0 virbr0
student@ubuntu:/etc$ ip route
default via 192.168.1.1 dev ens33 proto static metric 100
169.254.0.0/16 dev ens33 scope link metric 1000
192.168.1.0/24 dev ens33 proto kernel scope link src 192.168.1.24 metric 100
192.168.122.0/24 dev virbr0 proto kernel scope link src 192.168.122.1 linkdown
student@ubuntu:/etc$
```

Figure 35.7: Using **route** and **ip route**

For the exclusive use of LFS301 corp class taught 06 to 09 December

Default Route

- Used when no routing table entry is matched
- Can be obtained dynamically using DHCP
- Can be manually configured (static)
 - **Red Hat**-derived systems: GATEWAY=x.x.x.x in either
`/etc/sysconfig/network` or
`/etc/sysconfig/network-scripts/ifcfg-ethX`
 - **Debian**-derived systems: gateway=x.x.x.x in
`/etc/network/interfaces`
- Can use **nmcli** to avoid modifying files in `/etc` by hand:


```
$ sudo nmcli con mod virbr0 ipv4.routes 192.168.10.0/24 \
+ipv4.gateway 192.168.122.0
$ sudo nmcli con up virbr0
```

The **default route** is the way packets are sent when there is no other match in the routing table for reaching the specified network.

You can set the default gateway at run time with:

```
$ sudo route add default gw 192.168.1.10 enp2s0
$ route
```

Kernel IP routing table							
Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
default	192.168.1.10	0.0.0.0	UG	0	0	0	enp2s0
default	192.168.1.1	0.0.0.0	UG	1024	0	0	enp2s0
172.16.132.0	0.0.0.0	255.255.255.0	U	0	0	0	vmnet1
192.168.1.0	0.0.0.0	255.255.255.0	U	0	0	0	enp2s0
192.168.113.0	0.0.0.0	255.255.255.0	U	0	0	0	vmnet8

Note this might wipe out your network connection! You can restore either by resetting the network, or in the above example by doing:

```
$ sudo route add default gw 192.168.1.1 enp2s0
```

These changes are not persistent and will not survive a system restart.

Static Routes

- Used to control packet flow when there is more than one router or route
- Defined per interface
- Can be persistent or non-persistent
- Using **ip** (non-persistent)
- Or persistently:
 - **Red Hat**-based system:
`/etc/sysconfig/network-scripts/route-ethX`
 - **Debian**-based system:
`/etc/network/interfaces`
 - **SUSE**-based system:
`/etc/sysconfig/network/ifroute-eth0`
- Use **nmcli** to avoid changing files in `/etc` by hand (Exercise to follow)

When the system can access more than one route or router, or perhaps there are multiple interfaces, it is useful to selectively control which packets go to which.

Either the **route** or **ip** command can be used to set a non-persistent route as in:

```
$ sudo ip route add 10.5.0.0/16 via 192.168.1.100
```

On a **Red Hat**-based system, a persistent route can be set by editing `/etc/sysconfig/network-scripts/route-ethX` as shown by:

```
$ cat /etc/sysconfig/network-scripts/route-eth0
10.5.0.0/16 via 172.17.9.1
```

On a **Debian**-based system you need to add lines to `/etc/network/interfaces` such as:

```
iface eth1 inet dhcp
  post-up route add -host 10.1.2.51 eth1
  post-up route add -host 10.1.2.52 eth1
```

On a **SUSE**-based system you need to add to or create a file such as `/etc/sysconfig/network/ifroute-eth0` with lines like:

```
# Destination Gateway Netmask Interface [Type] [Options]
192.168.1.150 192.168.1.1 255.255.255.255 eth0
10.1.1.150 192.168.233.1.1 eth0
10.1.1.0/24 192.168.1.1 - eth0
```

where each field is separated by tabs.

For the exclusive use of LFS301 corp class taught 06 to 09 December

35.8 DNS and Name Resolution

DNS and Name Resolution

- Name resolution: converting a hostname to an IP address
 - Static name resolution
 - Dynamic name resolution (**DNS**)
- Reverse resolution: converting an IP address to a hostname
- From the command line:

```
$ dig linuxfoundation.org
$ host linuxfoundation.org
$ nslookup linuxfoundation.org
```

Name resolution is the act of translating hostnames to the IP addresses of their hosts.

For example, a browser or email client will take `training.linuxfoundation.org` and resolve the name to the IP address of the server (or servers) that serve `training.linuxfoundation.org` in order to transmit to and from that location.

This can be done in either **static** fashion, using `/etc/hosts`, or dynamically using **DNS** servers).

There are several command line tools that can be used to resolve the IP address of a hostname:

- **dig**: generates the most information and has many options
- **host**: more compact
- **nslookup**: older

dig is newest and the other are sometimes considered deprecated, but the output for **host** is easiest to read and contains the basic information.

One sometimes also requires **reverse resolution**: converting an IP address to a hostname. Try feeding these 3 utilities a known IP address instead of a hostname, and examine the output.

For the exclusive use of LFS301 corp class taught 06 to 09 December

/etc/hosts

- Used for local or static hostname resolution
- Useful for small isolated networks
- Usually checked before DNS
- Priority can be configured under `/etc/nsswitch.conf` but not often used today

```
student@ubuntu:/etc$ ls -l host*
-rw-r--r-- 1 root root 92 Oct 22 2015 host.conf
-rw-r--r-- 1 root root 7 Apr 21 08:46 hostname
-rw-r--r-- 1 root root 221 Apr 21 08:46 hosts
-rw-r--r-- 1 root root 411 Apr 20 17:14 hosts.allow
-rw-r--r-- 1 root root 711 Apr 20 17:14 hosts.deny
```

`/etc/hosts` holds a local database of hostnames and IP addresses. It contains a set of records (each taking one line) which map IP addresses with corresponding hostnames and aliases.

A typical `/etc/hosts` file looks like:



/etc/hosts

```
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.1.100 hans hans7 hans64
192.168.1.150 bethe bethe7 bethe64
192.168.1.2 hp-printer
192.168.1.10 test32 test64 oldpc
```

Such static name resolution is primarily used for local, small, isolated networks. It is generally checked before **DNS** is attempted to resolve an address; however, this priority can be controlled by `/etc/nsswitch.conf`, but is not often done today.

The other host-related files in `/etc` are:

`/etc/hosts.deny`, `/etc/hosts.allow`.

These are self documenting and their purpose is obvious from their names. The allow file is searched first and the deny file is only searched if the query is not found there.

`/etc/host.conf` contains general configuration information; it is rarely used.

For the exclusive use of LFS301 corp class taught 06 to 09 December

DNS

- Uses dynamic name servers
- `/etc/resolv.conf`
 - Can specify particular domains to search.
 - Defines a strict order of nameservers to query.
 - May be manually configured or updated from a service such as **DHCP** (Dynamic Host Configuration Protocol).

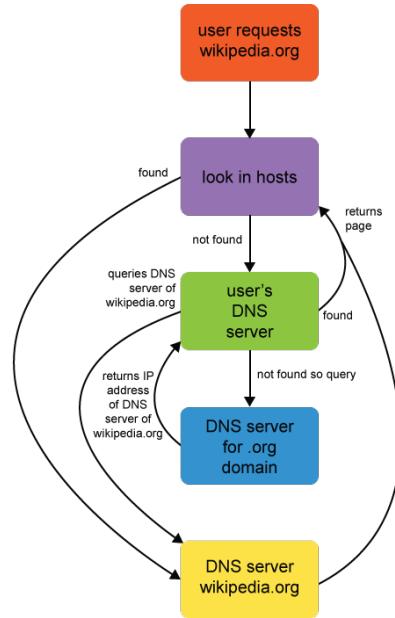


Figure 35.8: DNS

If name resolution cannot be done locally using `/etc/hosts` then the system will query a **DNS** (Domain Name Server) server.

DNS is dynamic and consists of a network of servers which a client uses to look up names. The service is distributed; any one **DNS** server has only information about its **zone of authority**; however, all of them together can cooperate to resolve any name.

The machine's usage of **DNS** is configured in `/etc/resolv.conf`, which historically has looked like:



/etc/resolv.conf

```
search example.com aps.org
nameserver 192.168.1.1
nameserver 8.8.8.8
```

However, most modern systems will have a `/etc/resolv.conf` file generated automatically, such as:

```
# Generated by NetworkManager
```

```
1 192.168.1.1
```

which was generated by **NetworkManager** invoking **DHCP** on the primary network interface.

For the exclusive use of LFS301 corp class taught 06 to 09 December

35.9 Network Diagnostics

Network Diagnostics

- **ping**
- **traceroute**
- **mtr**
- **dig**

A number of basic network utilities are in every system administrator's toolbox, including:

ping:

Sends 64-byte test packets to designated network hosts and tries to report back on the time required to reach it (in milliseconds), any lost packets, and some other parameters. Note that the exact output will vary according to the host being targeted, but at least you can see that the network is working and the host is reachable.

traceroute:

Is used to display a network path to a destination. It shows the routers packets flow through to get to a host, as well as the time it takes for each **hop**.

mtr:

Combines the functionality of **ping** and **traceroute** and creates a continuously updated display, like **top**.

dig:

Is useful for testing **DNS** functionality. Note one can also use **host** or **nslookup**, older programs that also try to return **DNS** information about a host.

Note that some **Linux** distributions, such as **RHEL**, may require root privilege (as with **sudo**) in order to run the first three diagnostic utilities.

Examples:

```
$ ping -c 10 linuxfoundation.org  
$ traceroute linuxfoundation.org  
$ mtr linuxfoundation.org
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

ping Example

```
[student@fedora ~]$ ping -c 10 linuxfoundation.org
PING linuxfoundation.org (23.185.0.4) 56(84) bytes of data.
64 bytes from 23.185.0.4 (23.185.0.4): icmp_seq=1 ttl=128 time=21.3 ms
64 bytes from 23.185.0.4 (23.185.0.4): icmp_seq=2 ttl=128 time=20.2 ms
64 bytes from 23.185.0.4 (23.185.0.4): icmp_seq=3 ttl=128 time=19.2 ms
64 bytes from 23.185.0.4 (23.185.0.4): icmp_seq=4 ttl=128 time=20.7 ms
64 bytes from 23.185.0.4 (23.185.0.4): icmp_seq=5 ttl=128 time=19.8 ms
64 bytes from 23.185.0.4 (23.185.0.4): icmp_seq=6 ttl=128 time=19.6 ms
64 bytes from 23.185.0.4 (23.185.0.4): icmp_seq=7 ttl=128 time=19.5 ms
64 bytes from 23.185.0.4 (23.185.0.4): icmp_seq=8 ttl=128 time=20.3 ms
64 bytes from 23.185.0.4 (23.185.0.4): icmp_seq=9 ttl=128 time=21.0 ms
64 bytes from 23.185.0.4 (23.185.0.4): icmp_seq=10 ttl=128 time=20.9 ms

--- linuxfoundation.org ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9012ms
rtt min/avg/max/mdev = 19.247/20.259/21.282/0.670 ms
[student@fedora ~]$
```

Figure 35.9: ping

traceroute Example

```
c8:/tmp>traceroute google.com
traceroute to google.com (172.217.9.78), 30 hops max, 60 byte packets
 1  router (192.168.1.1)  0.340 ms  0.425 ms  0.544 ms
 2  * * *
 3  096-034-031-180.biz.spectrum.com (96.34.31.180)  19.900 ms  19.553 ms  18.011 ms
 4  acr06ftbgwi-gbe-8-46.ftbg.wi.charter.com (96.34.30.244)  19.682 ms  20.144 ms  19.848 ms
 5  cts03alxnmn-tge-3-0-0.alxn.mn.charter.com (96.34.25.137)  26.237 ms  26.465 ms  26.214 ms
 6  crr01rochmn-bue-2.roch.mn.charter.com (96.34.25.166)  26.445 ms  23.845 ms  25.653 ms
 7  crr01stcdmn-bue-6.stcd.mn.charter.com (96.34.25.0)  27.146 ms  21.003 ms  28.613 ms
 8  bbr02slidla-tge-0-1-0-6.slid.la.charter.com (96.34.1.188)  35.569 ms  33.988 ms  31.121 ms
 9  bbr02chcgil-bue-1.chcg.il.charter.com (96.34.1.149)  46.788 ms  46.397 ms  39.472 ms
10  prr01chcgil-bue-4.chcg.il.charter.com (96.34.3.11)  44.474 ms  44.081 ms  43.731 ms
11  096-034-152-159.biz.spectrum.com (96.34.152.159)  42.068 ms 096-034-152-155.biz.spectrum.co
m (96.34.152.155)  42.878 ms 096-034-152-159.biz.spectrum.com (96.34.152.159)  44.781 ms
12  108.170.243.193 (108.170.243.193)  40.572 ms 108.170.243.174 (108.170.243.174)  39.112 ms 1
08.170.243.193 (108.170.243.193)  42.198 ms
13  72.14.239.123 (72.14.239.123)  42.209 ms  39.873 ms 72.14.232.192 (72.14.232.192)  45.645 m
s
14  72.14.239.123 (72.14.239.123)  43.850 ms 108.170.244.2 (108.170.244.2)  43.267 ms 108.170.2
44.15 (108.170.244.15)  42.362 ms
15  ord38s09-in-f14.1e100.net (172.217.9.78)  42.122 ms  41.888 ms  40.801 ms
c8:/tmp>
```

Figure 35.10: traceroute

mtr Example

My traceroute [v0.92]								2021-03-26T09:11:33-0500		
		Keys: Help Display mode		Restart statistics		Order of fields		quit		
Host	1. router	Packets				Pings				
		Loss%	Snt	Last	Avg	Best	Wrst	0.4	StDev	
2. ???		0.0%	76	0.3	0.3	0.2	0.4	0.0		
3. 096-034-031-180.biz.spectrum.com		0.0%	76	9.5	9.9	8.2	27.5	2.4		
4. acr06ftbgwi-gbe-8-46.ftbg.wi.charter.com		0.0%	76	19.9	11.5	8.8	21.8	3.3		
5. crr01onlswi-bue-402.onls.wi.charter.com		0.0%	76	12.8	14.0	11.9	31.9	3.0		
6. crr01euclwi-bue-400.eucl.wi.charter.com		0.0%	76	14.2	16.0	13.4	32.6	3.2		
7. bbr01euclwi-bue-100.eucl.wi.charter.com		0.0%	76	18.1	18.9	14.2	27.4	2.8		
8. prr01mplsmn-bue-801.mpls.mn.charter.com		0.0%	75	20.8	28.8	19.7	505.9	55.9		
9. 10ge5-18.core1.msp1.he.net		0.0%	75	20.9	26.3	19.3	44.5	7.6		
10. 100gell-2.core1.seal.he.net		0.0%	75	89.3	77.4	71.3	101.7	6.6		
11. 100ge15-1.core1.pdx1.he.net		0.0%	75	74.4	88.1	74.4	104.2	8.6		
12. infinity-internet-inc.gigabitethernet2-11.core1		0.0%	75	74.5	77.0	74.5	89.2	2.9		
13. xe-6-0-0.r-1.linuxfoundation.org		0.0%	75	77.1	77.0	74.5	88.1	3.0		
14. kernel.org		0.0%	75	74.8	79.3	74.2	94.1	5.2		

Figure 35.11: mtr

For the exclusive use of LFS301 corp class taught 06 to 09 December

35.10 Labs

Exercise 35.1: Static Configuration of a Network Interface



Please Note

You may have to use a different network interface name than eth0. You can most easily do this exercise with **nmtui** or your system's graphical interface. We will present a command line solution, but beware details may not exactly fit your distribution flavor or fashion.

1. Show your current IP address, default route and **DNS** settings for eth0. Keep a copy of them for resetting later.
2. Bring down eth0 and reconfigure to use a static address instead of **DCHP**, using the information you just recorded.
3. Bring the interface back up, and configure the nameserver resolver with the information that you noted before. Verify your hostname and then **ping** it.
4. Make sure your configuration works after a reboot.

You will probably want to restore your configuration when you are done.

Solution 35.1

```
1. $ ip addr show eth0
   $ ip route
   $ cp /etc/resolv.conf resolv.conf.keep
```

or

```
$ ifconfig eth0
$ route -n
$ cp /etc/resolv.conf resolv.conf.keep
```

```
2. $ sudo ip link set eth0 down
```

or

```
$ sudo ifconfig eth0 down
```



On RedHat / CentOS

Make sure the following is in `/etc/sysconfig/network-scripts/ifcfg-eth0` on **Red Hat**-based systems:



in `/etc/sysconfig/network-scripts/ifcfg-eth0`

```
DEVICE=eth0
BOOTPROTO=static
ONBOOT=yes
IPADDR=noted from step 1
NETMASK=noted from step 1
GATEWAY=noted from step 1
```

For the exclusive use of LFS301 corp class taught 06 to 09 December



On openSUSE , SLES, OpenSUSE, and Debian-based systems

On **SUSE**-based systems edit the file in `/etc/sysconfig/network` in the same way, and on **Debian**-based systems edit `/etc/networking/interfaces` to include:



in /etc/sysconfig/network or /etc/networking/interfaces

```
iface eth0 inet static
address noted from step 1
netmask noted from step 1
gateway noted from step 1
```

3. `$ sudo ip link set eth0 up`

or

```
$ sudo ifconfig eth0 up

$ sudo cp resolv.conf.keep /etc/resolv.conf
$ cat /etc/sysconfig/network
$ cat /etc/hosts
$ ping yourhostname
```

4. `$ sudo reboot`
`$ ping hostname`

✍ Exercise 35.2: Adding a Static Hostname

In this exercise we will add entries to the local host database.

1. Open `/etc/hosts` and add an entry for `mysystem.mydomain` that will point to the IP address associated with your network card.
2. Add a second entry that will make all references to `ad.doubleclick.net` point to `127.0.0.1`.
3. As an optional exercise, download the host file from: <http://winhelp2002.mvps.org/hosts2.htm> or more directly from <http://winhelp2002.mvps.org/hosts.txt>, and install it on your system. Do you notice any difference using your browser with and without the new host file in place?

✓ Solution 35.2

1. `$ sudo sh -c "echo 192.168.1.180 mysystem.mydomain >> /etc/hosts"`
`$ ping mysystem.mydomain`
2. `$ sudo sh -c "echo 127.0.0.1 ad.doubleclick.net >> /etc/hosts"`
`$ ping ad.doubleclick.net`
3. `$ wget http://winhelp2002.mvps.org/hosts.txt`

```

1  --2014-11-01 08:57:12--  http://winhelp2002.mvps.org/hosts.txt
2  Resolving winhelp2002.mvps.org (winhelp2002.mvps.org)... 216.155.126.40
3  Connecting to winhelp2002.mvps.org (winhelp2002.mvps.org)|216.155.126.40|:80... connected.
4  HTTP request sent, awaiting response... 200 OK
5  Length: 514744 (503K) [text/plain]
6  Saving to: hosts.txt
7
8  100%[=====] 514,744      977KB/s   in 0.5s
9
10 2014-11-01 08:57:13 (977 KB/s) - hosts.txt saved [514744/514744]
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

```
$ sudo sh -c "cat hosts.txt >> /etc/hosts"
```

Exercise 35.3: Adding a Network Interface Alias/Address using nmcli

We are going to add an additional IPv4 address to your system and make it persistent. We will do this without editing files under `/dev` directly, using **nmcli**.

1. First obtain your current internet address and interface name:

```
$ sudo nmcli con
1 NAME           UUID                               TYPE      DEVICE
2 Auto Ethernet  1c46bf37-2e4c-460d-8b20-421540f7d0e2 802-3-ethernet  ens33
3 virbr0         a84a332f-38e3-445a-a377-4363a8eb963f bridge      virbr0
```

shows the name of the connection is Auto Ethernet.

```
$ sudo nmcli con show "Auto Ethernet" | grep IP4.ADDRESS
1 IP4.ADDRESS[1]:                         172.16.2.135/24
```

shows the address as 172.16.2.135 Note that this command shows all information about the connection and you could have specified the UUID instead of the NAME as in:

```
$ nmcli con show 1c46bf37-2e4c-460d-8b20-421540f7d0e2
```

2. Add a new address that your machine can be seen by:

```
$ sudo nmcli con modify "Auto Ethernet" +ipv4.addresses 172.16.2.140/24
```

3. Activate it and test to see if it is there:

```
$ sudo nmcli con up "Auto Ethernet"
1 Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection/6)
2
3 $ ping -c 3 172.16.2.140
4 PING 172.16.2.140 (172.16.2.140) 56(84) bytes of data.
5 64 bytes from 172.16.2.140: icmp_seq=1 ttl=64 time=0.038 ms
6 64 bytes from 172.16.2.140: icmp_seq=2 ttl=64 time=0.034 ms
7 64 bytes from 172.16.2.140: icmp_seq=3 ttl=64 time=0.032 ms
8
9 --- 172.16.2.140 ping statistics ---
10 3 packets transmitted, 3 received, 0% packet loss, time 1998ms
11 rtt min/avg/max/mdev = 0.032/0.034/0.038/0.007 ms
```

4. Clean up by removing the alias:

```
$ sudo nmcli con modify "Auto Ethernet" -ipv4.addresses 172.16.2.140/24
...
$ sudo nmcli con up "Auto Ethernet"
...
```

Exercise 35.4: Adding a Static Route using nmcli

We are going to add a static IPv4 route address to your system and make it persistent. We will do this without editing files under `/dev` directly, using **nmcli**.

1. Begin by examining your current routing tables, using both **route** and **ip**:

```
$ route
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

```

1 Kernel IP routing table
2 Destination      Gateway        Genmask        Flags Metric Ref  Use Iface
3 default          172.16.2.2    0.0.0.0       UG    100    0      0 ens33
4 link-local       *              255.255.0.0   U     1000   0      0 ens33
5 172.16.2.0      *              255.255.255.0 U     100    0      0 ens33
6 192.168.122.0   *              255.255.255.0 U     0      0      0 virbr0

```

```

$ ip route
1 default via 172.16.2.2 dev ens33 proto static metric 100
2 169.254.0.0/16 dev ens33 scope link metric 1000
3 172.16.2.0/24 dev ens33 proto kernel scope link src 172.16.2.135 metric 100
4 192.168.122.0/24 dev virbr0 proto kernel scope link src 192.168.122.1 linkdown

```

2. Add a new route using **nmcli**:

```
$ sudo nmcli conn mod "Auto Ethernet" +ipv4.routes "192.168.100.0/24 172.16.2.1"
```

3. Note it has not yet taken effect:

```

$ route
1 Kernel IP routing table
2 Destination      Gateway        Genmask        Flags Metric Ref  Use Iface
3 default          172.16.2.2    0.0.0.0       UG    100    0      0 ens33
4 link-local       *              255.255.0.0   U     1000   0      0 ens33
5 172.16.2.0      *              255.255.255.0 U     100    0      0 ens33
6 192.168.122.0   *              255.255.255.0 U     0      0      0 virbr0

```

4. Reload the interface to have it take effect and show it has:

```
$ sudo nmcli conn up "Auto Ethernet"
```

```
1 Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection/25)
```

```
$ route
```

```

1 Kernel IP routing table
2 Destination      Gateway        Genmask        Flags Metric Ref  Use Iface
3 default          172.16.2.2    0.0.0.0       UG    100    0      0 ens33
4 link-local       *              255.255.0.0   U     1000   0      0 ens33
5 172.16.2.0      *              255.255.255.0 U     100    0      0 ens33
6 192.168.100.0   172.16.2.1   255.255.255.0 UG    100    0      0 ens33
7 192.168.122.0   *              255.255.255.0 U     0      0      0 virbr0

```

5. Reboot and verify the route has taken effect (i.e., it is **persistent**: If so remove it:

```
$ route
```

```

1 Kernel IP routing table
2 Destination      Gateway        Genmask        Flags Metric Ref  Use Iface
3 default          172.16.2.2    0.0.0.0       UG    100    0      0 ens33
4 link-local       *              255.255.0.0   U     1000   0      0 ens33
5 172.16.2.0      *              255.255.255.0 U     100    0      0 ens33
6 192.168.100.0   172.16.2.1   255.255.255.0 UG    100    0      0 ens33
7 192.168.122.0   *              255.255.255.0 U     0      0      0 virbr0

```

```
$ sudo nmcli conn mod "Auto Ethernet" -ipv4.routes "192.168.100.0/24 172.16.2.1"
```

```
$ sudo nmcli conn up "Auto Ethernet"
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

```
1 Connection successfully activated (D-Bus active path: /org/freedesktop/NetworkManager/ActiveConnection/3)
2 $ route
3 Kernel IP routing table
4 Destination      Gateway          Genmask        Flags Metric Ref  Use Iface
5 default         172.16.2.2      0.0.0.0       UG    100    0      0 ens33
6 link-local      *               255.255.0.0   U     1000   0      0 ens33
7 172.16.2.0      *               255.255.255.0 U     100    0      0 ens33
8 192.168.122.0   *               255.255.255.0 U     0      0      0 virbr0
```

6. Note you can set a route with either **route** or **ip** from the command line but it won't survive a reboot as in:

```
$ sudo ip route add 192.168.100.0/24 via 172.16.2.1
$ sudo route
....
```

You can verify that a route established this way is not persistent.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Chapter 36

Firewalls



36.1	Firewalls	450
36.2	Interfaces	453
36.3	firewalld	455
36.4	Zones	457
36.5	Source Management	461
36.6	Service and Port Management	462
36.7	Labs	464

36.1 Firewalls

What is a Firewall?

- Essential security facility in modern networks
- Control incoming and outgoing access to systems and network
- Protect against intrusions and other attack vectors
- Control trust level on particular interfaces and/or addresses
- **Rules** control flexible barriers depending on trust levels and network topography

A **firewall** is a network security system that monitors and controls all network traffic. It applies **rules** on both incoming and outgoing network connections and packets and builds flexible barriers (i.e., firewalls) depending on the level of trust of a given connection.

Firewalls can be hardware or software based. They are found both in network routers as well as in individual computers, or network nodes. Many firewalls also have routing capabilities.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Packet Filtering

- Basis for building firewalls
- Each packet in a network stream is examined
- System-defined **rules** are followed
- Actions can include, accept, drop, mangle, redirect, etc.

Almost all firewalls are based on **Packet Filtering**

Information is transmitted across networks in the form of packets, each one of which has:

- Header
- Payload
- Footer

The header and footer contain information about destination and source addresses, what kind of packet it is and which protocol it obeys, various flags, and which packet number this is in a stream, and all sorts of other metadata about transmissions

The actual data is in the payload.

Packet filtering intercepts packets at one or more stages in network transmission, including application, transport, network and datalink.

A firewall establishes a set of **rules** by which each packet may be :

- Accepted or Rejected based on content, address etc.
- Mangled in some way
- Redirected to another address
- Inspected for security reasons
- etc.

Various utilities exist for establishing rules and actions to be taken as the result of packet filtering.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Firewall Generations

- **Packet Filtering:** Inspect each packet and drop, reject or send on
- **Stateful Filters:** Examine **connection state** and consider if part of a new connection or an existing one, or none
- **Application Layer Firewalls:** Aware of application and protocol packet is associated with

Early firewalls (dating back to the late 1980's) were based on **packet filtering**; the content of each network packet was inspected and was either dropped, rejected, or sent on. No consideration was given about the **connection state**; what stream of traffic the packet was part of.

The next generation of firewalls were based on **stateful filters** which also examine the **connection state** of the packet; is it a new connection, part of an already existing one, or part of none. Denial of service attacks can bombard this kind of firewall to try and overwhelm it.

The third generation of firewalls is called **Application Layer Firewalls**, and are aware of the kind of application and protocol the connection is using. They can block anything which should not be part of the normal flow.

For the exclusive use of LFS301 corp class taught 06 to 09 December

36.2 Interfaces

Firewall Interfaces and Tools

- Command line tools:
 - **iptables**
 - **firewall-cmd**
 - **ufw**
- Graphical interfaces
 - **system-config-firewall**
 - **firewall-config**
 - **gufw**
 - **yast**
- Other utilities, such as **shorewall**, exist
- Command line tools vary less between distributions

Configuring your system's firewall can be done by:

- Using relatively low-level tools from the command line, combined with editing various configuration files in the `/etc` subdirectory tree: **iptables**, **firewall-cmd**, **ufw**, etc.
- Using robust graphical interfaces: **system-config-firewall**, **firewall-config**, **gufw**, **yast**, etc.

We will work with the lower level tools for the following reasons:

- They change less often than the graphical ones.
- They tend to have a larger set of capabilities.
- They vary little from distribution to distribution, while the **GUIs** tend to be quite different and each confined to only one family of distributions.

The disadvantage is they can seem more difficult to learn at first.

In the following we will concentrate on the use of the modern **firewalld** package, which includes both **firewall-cmd** and **firewall-config**. For distributions which do not have it by default, it can be installed from source rather easily, as we will do if necessary in an exercise.

Why we are not working with iptables

- Most firewall installations today are actually using the **iptables** package on the user side.
- This currently interfaces the same kernel firewall implementation code as **firewalld** which we will discuss more.
- We have decided not to teach **iptables** because it requires much more time to get to useful functionality
- However, **iptables** is discussed in detail in the next course in the **Linux Foundation** system administrator sequence:
LFS311: Linux for System Engineers
- See <https://training.linuxfoundation.org/linux-courses/system-administration-training/linux-for-system-engineers>

For the exclusive use of LFS301 corp class taught 06 to 09 December

36.3 firewalld

firewalld and firewall-cmd

- Dynamic Firewall Manager (**firewalld**)
- Works with **zones**
- Supports **IPv4** and **IPv6**
- Separates **Runtime** and **Permanent** configurations
- Main tool is **firewall-cmd**
- Replaces **iptables** utility
- Part of **systemd** universe
- Gradually being adopted by major distributions, can also be installed from source

firewalld is the **Dynamic Firewall Manager**. It utilizes network/firewall **zones** which have defined levels of trust for network interfaces or connections. It supports both **IPv4** and **IPv6** protocols.

In addition, it separates **runtime** and **permanent** (persistent) changes to configuration, and also includes interfaces for services or applications to add firewall rules.

Configuration files are kept in `/etc/firewalld` and `/usr/lib/firewalld`; the files in `/etc/firewalld` override those in the other directory and are the ones a system administrator should work on.

The command line tool is actually **firewall-cmd** which we will discuss.

We recommend that before getting any further, you run:

```
$ firewall-cmd --help
1 Usage: firewall-cmd [OPTIONS...]
2 ....
3 Status Options
4   --state           Return and print firewalld state
5   --reload          Reload firewall and keep state information
6   --complete-reload Reload firewall and loose state information
7   --runtime-to-permanent Create permanent from runtime configuration
8 ....
```

which runs about 200 lines, so it is too long to include here. However, you will see that almost all options are rather obvious as they are well named.

As a service, **firewalld** replaces the older **iptables**. It is an error to run both services at the same time.

For the exclusive use of LFS301 corp class taught 06 to 09 December

firewalld Service Status

- **firewalld** service must be enabled/disabled, start/stopped like any other service:

```
$ sudo systemctl [enable/disable] firewalld  
$ sudo systemctl [start/stop] firewalld
```

- Show status:

```
$ sudo systemctl status firewalld  
$ sudo firewall-cmd --state
```

- With more than one network interface, must turn on **IP Forwarding**:

```
$ sudo sysctl net.ipv4.ip_forward=1  
$ echo 1 > /proc/sys/net/ipv4/ip_forward
```

firewalld is a service which needs to be running to use and configure the firewall, and is enabled/disabled, or started or stopped in the usual ways as shown above. You can show the current state in either of the following ways:

```
$ sudo systemctl status firewalld
```

```
1 firewalld.service - firewalld - dynamic firewall daemon  
2   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled)  
3     Active: active (running) since Tue 2015-04-28 12:00:59 CDT; 5min ago  
4       Main PID: 777 (firewalld)  
5         ...
```

```
$ sudo firewall-cmd --state
```

running

Note that if you have more than one network interface when using **IPv4**, you have to turn on **ip forwarding**. You can do this at run time by doing either of:

```
$ sudo sysctl net.ipv4.ip_forward=1  
$ echo 1 > /proc/sys/net/ipv4/ip_forward
```

(where the second command has to be run as root to get the **echo** to work properly.) However this is not persistent. To do that you have to add the line **net.ipv4.ip_forward=1** to **/etc/sysctl.conf**, and then reboot or type **sudo sysctl -p** to read in the new setting without rebooting.

36.4 Zones

Zones

- Each zone has defined trust level and packet management behavior:
 - **drop** All incoming packets dropped
 - **block** All incoming network connections rejected
 - **public** By default trust none
 - **external** public with masquerading
 - **dmz** (Demilitarized Zone) Allow some services to some connections
 - **work** Trust (somewhat) certain network connections.
 - **home** Trust (mostly) certain network connections.
 - **internal** Similar to the **work** zone.
 - **trusted** All connections allowed.

firewalld works with **zones**, each of which has a defined level of trust and a certain known behavior for incoming and outgoing packets. Each interface belongs to a particular zone (normally it is **NetworkManager** which informs **firewalld** which zone is applicable), but this can be changed with **firewall-cmd** or the **firewall-config** GUI. The zones are:

- **drop**: All incoming packets are dropped with no reply. Only outgoing connections are permitted.
- **block**: All incoming network connections are rejected. The only permitted connections are those from within the system.
- **public**: Do not trust any computers on the network; only certain consciously selected incoming connections are permitted.
- **external**: Used when **masquerading** is being used, such as in routers. Trust levels the same as in **public**.
- **dmz**: (Demilitarized Zone) Used when access to some (but not all) services are to be allowed to the public. Only particular incoming connections are allowed.
- **work**: Trust (but not completely) connected nodes to be not harmful. Only certain incoming connections are allowed.
- **home**: You mostly trust the other network nodes, but still select which incoming connections are allowed.
- **internal**: Similar to the **work** zone.
- **trusted**: All network connections are allowed.

On system installation, most if not all **Linux** distributions will select the **public** zone as default for all interfaces. The differences between some of the above zones are not obvious and we do not need to go into that much detail here, but note that one should not use a more open zone than necessary.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Zone Management Examples I

- Get the default zone:

```
$ sudo firewall-cmd --get-default-zone
public
```

- Obtain a list of zones currently being used:

```
$ sudo firewall-cmd --get-active-zones
public
    interfaces: eno1677736
```

- List all available zones:

```
$ sudo firewall-cmd --get-zones
block dmz drop external home internal public trusted work
```

```
$ firewall-cmd --help
```

```
1 ....
2 Zone Options
3   --get-default-zone  Print default zone for connections and interfaces
4   --set-default-zone=<zone>
5           Set default zone
6   --get-active-zones  Print currently active zones
7   --get-zones        Print predefined zones [P]
8   --get-services     Print predefined services [P]
9   --get-icmptypes   Print predefined icmptypes [P]
10  --get-zone-of-interface=<interface>
11      Print name of the zone the interface is bound to [P]
12  --get-zone-of-source=<source>[/<mask>]
13      Print name of the zone the source[/mask] is bound to [P]
14  --list-all-zones  List everything added for or enabled in all zones [P]
15  --new-zone=<zone>  Add a new zone [P only]
16  --delete-zone=<zone> Delete an existing zone [P only]
17  --zone=<zone>     Use this zone to set or query options, else default zone
18          Usable for options maked with [Z]
19  --get-target       Get the zone target [P] [Z]
20  --set-target=<target>
21          Set the zone target [P] [Z]
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

Zone Management Examples II

- To change the default zone to **trusted** and then change it back:

```
$ sudo firewall-cmd --set-default-zone=trusted
```

```
success
```

```
$ sudo firewall-cmd --set-default-zone=public
```

```
success
```

- To assign an interface temporarily to a particular zone:

```
$ sudo firewall-cmd --zone=internal --change-interface=en01
```

```
success
```

- To do so permanently:

```
$ sudo firewall-cmd --permanent --zone=internal --change-interface=en01
```

```
success
```

which creates the file `/etc/firewalld/zones/internal.xml`.

All work at controlling **firewalld** is done through the **firewall-cmd** program. More detailed information can be obtained with:

```
man firewalld-cmd
```

Zone Management Examples III

- To ascertain the zone associated with a particular interface:

```
$ sudo firewall-cmd --get-zone-of-interface=eno1  
public
```

- To get all details about a particular zone:

```
$ sudo firewall-cmd --zone=public --list-all  
public (active)  
  target: default  
  icmp-block-inversion: no  
  interfaces: eno1  
  sources:  
    services: chromecast libvirt libvirt-tls nfs nfs3 rsyncd ssh  
  ports:  
  protocols:  
  masquerade: no  
  forward-ports:  
  source-ports:  
  icmp-blocks:  
  rich rules:
```

36.5 Source Management

Source Management

- Bind zones to particular network addresses, not just interfaces
- Packets not from associated addresses go to default zone
- Assign with **firewall-cmd**:

```
$ sudo firewall-cmd --permanent --zone=trusted --add-source=192.168.1.0/24  
success
```

Any zone can be bound not just to a network interface, but also to particular network addresses. A packet is associated with a zone if:

- It comes from a source address already bound to the zone; or if not:
- It comes from an interface bound to the zone.

Any packet not fitting the above criteria is assigned to the default zone (i.e., usually **public**).

To assign a source to a zone (permanently):

```
$ sudo firewall-cmd --permanent --zone=trusted --add-source=192.168.1.0/24  
1 success
```

This says anyone with an IP address of 192.168.1.x will be added to the **trusted** zone.

Note you can remove a previously assigned source from a zone by using the **--remove-source** option, or change the zone by using **--change-source**.

You can list the sources bound to a zone with:

```
$ sudo firewall-cmd --permanent --zone=trusted --list-sources 192.168.1.0/24
```

In both of the above commands, if you leave out the **--permanent** option, you get only the current run time behavior.

For the exclusive use of LFS301 corp class taught 06 to 09 December

36.6 Service and Port Management

Service Management

- See available services:

```
$ sudo firewall-cmd --get-services
```

- See those in a zone:

```
$ sudo firewall-cmd --list-services --zone=public
```

- Add a service to a zone:

```
$ sudo firewall-cmd --permanent --zone=home --add-service=dhcp
success

$ sudo firewall-cmd --reload
```

So far we have assigned particular interfaces and/or addresses to zones, but we have not delineated what **services** and **ports** should be accessible within a zone. To see all the services available:

```
$ sudo firewall-cmd --get-services
1 RH-Satellite-6 amanda-client bacula bacula-client dhcp dhcpcv6 dhcpcv6-client dns ftp high-availability http\
2 https imaps ipp ipp-client ipsec kerberos kpasswd ldap ldaps libvirt libvirt-tls mdns mountd ms-wbt\
3 mysql nfs ntp openvpn pmcd pmproxy pmwebapi pmwebapis pop3s postgresql proxy-dhcp radius rpc-bind samba\
4 samba-client smtp ssh telnet tftp tftp-client transmission-client vnc-server wbem-https
```

or to see those currently accessible in a particular zone:

```
$ sudo firewall-cmd --list-services --zone=public
```

```
1 dhcpcv6-client ssh
```

To add a service to a zone:

```
$ sudo firewall-cmd --permanent --zone=home --add-service=dhcp
```

```
1 success
```

```
$ sudo firewall-cmd --reload
```

The second command, with `--reload`, is needed to make the change effective. It is also possible to add new services by editing the files in `/etc/firewalld/services`.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Port Management

- Port management:

```
$ sudo firewall-cmd --zone=home --list-ports  
$ sudo firewall-cmd --zone=home --add-port=21/tcp
```

Port management is very similar to service management:

```
$ sudo firewall-cmd --zone=home --add-port=21/tcp
```

```
1 success
```

```
$ sudo firewall-cmd --zone=home --list-ports
```

```
1 21/tcp
```

where by looking at `/etc/services` we can ascertain that port 21 corresponds to **ftp**:

```
$ grep " 21/tcp" /etc/services
```

```
1 ftp 21/tcp
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

36.7 Labs

Exercise 36.1: Installing firewalld

While most **Linux** distributions now have the **firewalld** package (which includes the **firewall-cmd** multi-purpose utility) available, it might not be installed on your system.

First you should check to see if it is already installed, with

```
$ which firewalld firewall-cmd
```

```
1 /usr/sbin/firewalld
2 /usr/bin/firewall-cmd
```

If you fail to find the program, then you need to install with one of the following, depending on your distribution in the usual way:

```
$ sudo dnf install firewalld
$ sudo zypper install firewalld
$ sudo apt-get install firewalld
```

If this fails, the **firewalld** package is not available for your distribution. In this case you will have to install from source.

To do this, go to <https://fedorahosted.org/firewalld/> and you can get the **git** source repository, or you can easily download the most recent tarball.

Then you have to follow the common procedure for installing from source (using whatever the current version is):

```
$ tar xvf firewalld-0.3.13.tar.bz2
$ cd firewalld-0.3.13
$ ./configure
$ make
$ sudo make install
```

Note this source also has an **uninstall** target:

```
$ sudo make uninstall
```

in case you have regrets.

You will have to deal with any inadequacies that come up in the **./configure** step, such as missing libraries etc. When you install from a packaging system, the distribution takes care of this for you, but from source it can be problematic. If you have run the **Linux Foundation's ready-for.sh** script on your system, you are unlikely to have problems.

Exercise 36.2: Examining firewall-cmd

We have only scratched the surface of how you can use the **firewalld** package. Almost everything is done by deploying **firewall-cmd** which is empowered to do a large variety of tasks, using options with very clear names.

To get a sense of this, there is really no substitute for just doing:

```
$ firewall-cmd --help
```

```
1 Usage: firewall-cmd [OPTIONS...]
2 ....
3 Service Options
4   --new-service=<service>
5       Add a new service [P only]
6   --delete-service=<service>
7       Delete and existing service [P only]
8 ....
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

which we will not reproduce here as it is 409 lines on an **RHEL 8** system.

For more detailed explanation of anything which piques your interest, do **man firewall-cmd** which explains things more deeply, and **man firewalld** which gives an overview, as well as a listing of other **man** pages that describe the various configuration files in **/etc**, and elucidate concepts such as **zones** and **services**.

Exercise 36.3: Adding Services to a Zone

Add the **http** and **https** services to the **public zone** and verify that they are currently listed.

Solution 36.3

```
$ sudo firewall-cmd --zone=public --add-service=http
```

```
1 success
```

```
$ sudo firewall-cmd --zone=public --add-service=https
```

```
1 success
```

```
$ sudo firewall-cmd --list-services --zone=public
```

```
1 dhcpcv6-client http https ssh
```

Note if you had run

```
$ sudo firewall-cmd --reload
$ sudo firewall-cmd --list-services --zone=public
```

```
1 dhcpcv6-client ssh
```

after adding the new services, they would disappear from the list! This curious behavior is because we did not include the **--permanent** flag when adding the services, and the **--reload** option reloads the known persistent services only.

Exercise 36.4: Using the firewall GUI

Each distribution has its own graphical interface for firewall administration. On **Red Hat**-based systems you can run **firewall-config**, on **Ubuntu** it is called **gufw**, and on **openSUSE** you can find it as part of **yast** on the graphical menu system.

We have concentrated on the command line approach simply because we want to be distribution-flexible. However, for most relatively simple firewall configuration tasks, you can probably do them efficiently with less memorization from the GUI.

Once you launch the firewall configuration GUI, do the previous exercise of adding **http** and **https** to the **public** zone, and verify that it has taken effect.

Make sure you take the time to understand the graphical interface.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Chapter 37

System Startup and Shutdown



37.1	Understanding the Boot Sequence	468
37.2	Boot Loaders	470
37.3	System Configuration Files in /etc	471
37.4	Shutting Down and Rebooting	474
37.5	Labs	475

37.1 Understanding the Boot Sequence

Boot Sequence

1. The **BIOS/UEFI** locates and executes the boot program, or boot loader.
2. The boot loader loads the kernel.
3. The kernel starts the **init** process (`pid=1`).
4. **init** manages system initialization, using **systemd** or the older **Upstart** and **SysVinit** startup scripts.

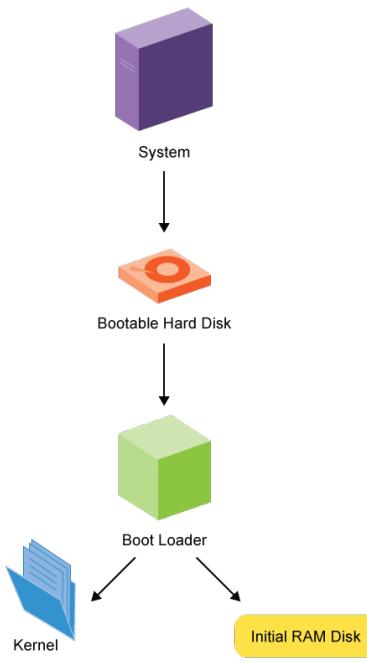


Figure 37.1: Boot Sequence

When power is applied to the computer, the computer can only perform the operations the **BIOS** (Basic Input Output System) orders it to do.

First, the **BIOS** runs the **POST** (Power On Self Test) which checks memory and hardware and then searches a specific location or device for a boot program. Typically, the boot program is found in the device's **MBR** (Master Boot Record), or using **UEFI**. Control of the computer is then transferred to this boot program (usually **GRUB**).

The boot program then loads the kernel into memory and executes it. On **x86** platforms (and many others) the kernel first has to decompress itself in place. It then performs hardware checks, gains access to important peripheral hardware, and eventually runs the **init** process. The **init** process in turn continues the system startup, running **init** scripts if **SysVinit** is being used, or managing either **Upstart** or **systemd**.

Newer computers are moving to **UEFI**, a replacement for the **BIOS**, which performs many of the same functions.

For the exclusive use of LFS301 corp class taught 06 to 09 December

BIOS

- Typically a **ROM** chip
- Contains code to control keyboard, display, disks
- During boot up, loads the boot loader

On the **x86** architecture, the **BIOS** contains all the code required to gain initial access to the keyboard, display screen, disk drives, serial communications, and a number of miscellaneous functions. Once the full system is running, most of these devices will have enhanced capabilities when complete and specialized device drivers can be loaded and take over.

The **BIOS** is typically placed in a **ROM** chip that comes with the computer (it is often called a **ROM BIOS**). This ensures that the **BIOS** will always be available and will not be damaged by disk failures. It also makes it possible for a computer to boot itself.

During the boot process, the **BIOS** loads the boot loader.

For the exclusive use of LFS301 corp class taught 06 to 09 December

37.2 Boot Loaders

Boot Loaders

- **GRUB**
- **efibootmgr**
- **LILO**
- **Das U-Boot**

Virtually all (non-embedded) modern **Linux** distributions use **GRUB** (GRand Unified Boot Loader). **GRUB**'s features include the ability to boot multiple operating systems, both a graphical and text based interface allowing ease of use over a serial cable, a powerful command line interface for interactive configuration, network-based diskless booting, and other advanced features.

efibootmgr is not actually a boot loader, but is a **boot manager**, used in conjunction with **GRUB** on multi-boot EFI systems.

The Linux Loader (**LILO**) is old and obsolete.

Das U-Boot is in wide usage for embedded systems. There are some other boot loaders used, including **bareboot**. However, we are not focused on embedded systems in the present course.

For the exclusive use of LFS301 corp class taught 06 to 09 December

37.3 System Configuration Files in /etc

Configuration Files in /etc

- **Linux** distributions cooperate to place certain kinds of files in standard places
- System-wide configuration files are generally placed in `/etc` subdirectories
- User-specific ones are in individual home directories
- All **Red Hat**-derived systems make extensive use of `/etc/sysconfig`
- All **Debian**-based systems use `/etc/default`
- **RHEL** and **SUSE** use both

The general rule about saving configuration information is:

- System wide files go in `/etc`.
- User-specific files go in `/home/[username]`.

This is not completely true; for example, default configuration information might be stored in `/usr/lib/systemd`, but can be overridden by files in `/etc/systemd`.

There should be only text files found under `/etc/`, no binary formats or data.

For the exclusive use of LFS301 corp class taught 06 to 09 December

/etc/sysconfig

- Files in this directory and its subdirectories are used by many system utilities service
- They are often consulted or invoked when the system starts and stops services or queries their status.
- For example, on one particular **RHEL** system:

```
File Edit View Search Terminal Help
c7:/tmp>cat /etc/sysconfig/selinux

# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - No SELinux policy is loaded.
#SELINUX=enforcing
SELINUX=disabled
# SELINUXTYPE= can take one of these two values:
#       targeted - Targeted processes are protected,
#       minimum - Modification of targeted policy. Only selected processes are protected.
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted
c7:/tmp|
```

Figure 37.2: Configuration Example: /etc/sysconfig/selinux

On a **RHEL** system:

```
c8:/tmp>ls /etc/sysconfig
anaconda      iptables-config  pmcd          saslauthd
atd           irqbalance     pmie_timers   selinux
cbq           kdump          pmlogger      smartmontools
chronyd       kernel         pmlogger_timers sshd
collectl      ksm            pmproxy       svnserve
console       libvirtd       qemu-ga      sysstat
cpupower      libvirt-guests radvd        sysstat.ioconf
crond         lm_sensors    raid-check   trace-cmd.conf
ebtables-config man-db        rhn          virtlockd
firewalld     modules       rpcbind      virtlogd
grub          network       rsyslog      wpa_supplicant
htcacheclean  network-scripts run-parts
ip6tables-config nftables.conf samba
c8:/tmp|
```

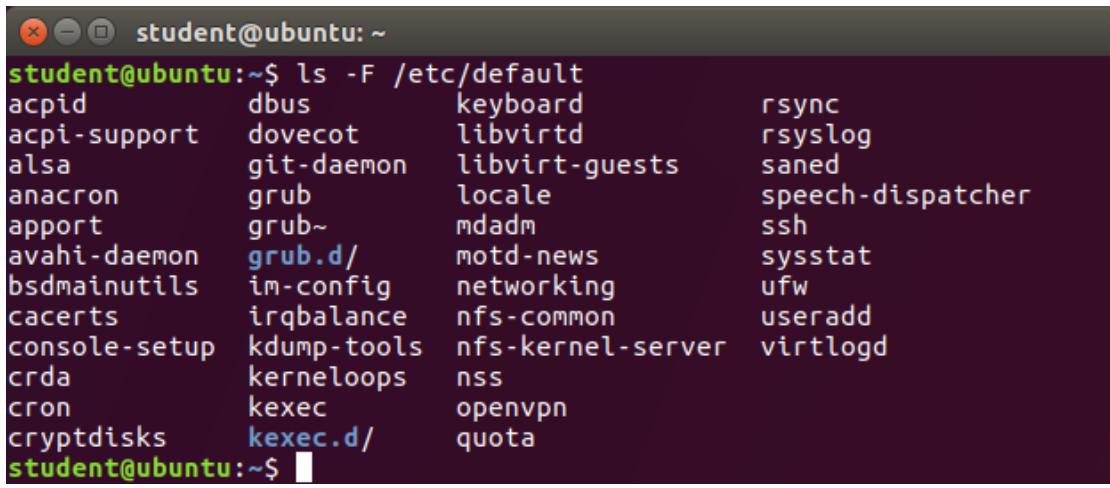
Figure 37.3: /etc/sysconfig on RHEL 8

For the exclusive use of LFS301 corp class taught 06 to 09 December

/etc/default

- Use is similar to that of **Red Hat's** [/etc/sysconfig](#)
- Files provide extra options when starting a service.
- Contain code to set environment variables.
- For example, [/etc/default/useradd](#) sets defaults used when creating new user accounts.
- All significant **Linux** distributions, including **Red Hat**-based ones, now have [/etc/default](#), even if they still have [/etc/sysconfig](#).

On an **Ubuntu** system:



```
student@ubuntu:~$ ls -F /etc/default
acpid      dbus      keyboard      rsync
acpi-support dovecot   libvirtd      rsyslog
alsa        git-daemon libvirt-guests saned
anacron     grub      locale       speech-dispatcher
apport      grub~    mdadm        ssh
avahi-daemon grub.d/  motd-news    sysstat
bsdmainutils im-config networking   ufw
cacerts     irqbalance nfs-common   useradd
console-setup kdump-tools nfs-kernel-server virtlogd
crda        kerneloops nss
cron        kexec
cryptdisks  kexec.d/  quota
```

Figure 37.4: /etc/default

For the exclusive use of LFS301 corp class taught 06 to 09 December

37.4 Shutting Down and Rebooting

Shutting Down and Rebooting

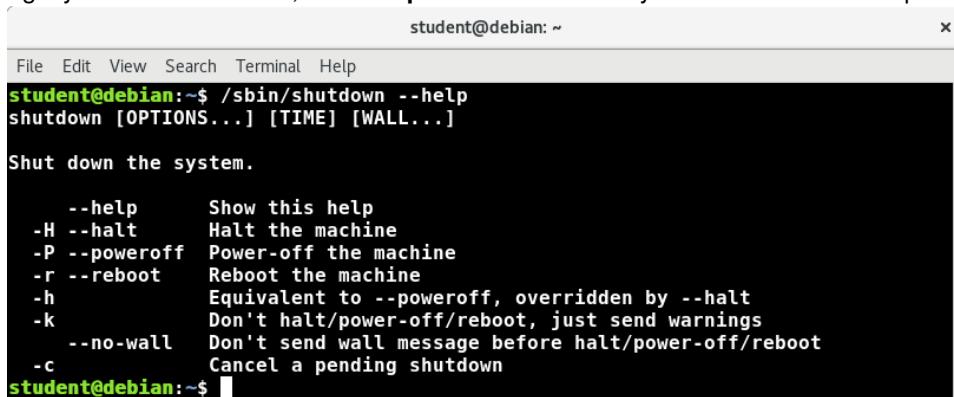
- **shutdown:**
 - Brings system down in a secure way
 - All users notified of pending action
 - Can be halted or rebooted
 - Without options, the default behaviour is complete power off
 - * Some distributions, such as **Ubuntu**, go to single user mode instead.
- Legacy commands such as **reboot**, **halt** and **poweroff** are still frequently used.

shutdown is used to bring the system down in a secure fashion, notifying all users that the system is going down and then stopping it in a graceful and non-destructive way. After it is shutdown the system is either halted or rebooted.

Examples:

```
$ shutdown -h +1 "Power Failure imminent"
$ shutdown -h now
$ shutdown -r now
$ shutdown now
```

There are also the legacy commands **reboot**, **halt** and **poweroff** which many veteran users use frequently.



The screenshot shows a terminal window with the following content:

```
student@debian:~$ /sbin/shutdown --help
shutdown [OPTIONS...] [TIME] [WALL...]

Shut down the system.

  --help      Show this help
  -H --halt   Halt the machine
  -P --poweroff Power-off the machine
  -r --reboot  Reboot the machine
  -h           Equivalent to --poweroff, overridden by --halt
  -k           Don't halt/power-off/reboot, just send warnings
  --no-wall   Don't send wall message before halt/power-off/reboot
  -c           Cancel a pending shutdown
student@debian:~$
```

Figure 37.5: **shutdown** Command

For the exclusive use of LFS301 corp class taught 06 to 09 December

37.5 Labs

📝 Exercise 37.1: Shutdown VS. Halt VS. Reboot

NOTE: This exercise requires that it be run from the console (i.e., not over the network through SSH).

1. Reboot the system using **shutdown**.
2. Power off the system using **shutdown**.
3. Power the system back up.

✔ Solution 37.1

1. `$ sudo shutdown -r now`
2. `$ sudo shutdown -h now`
3. Press the power button, or restart your virtual machine.

For the exclusive use of LFS301 corp class taught 06 to 09 December

Chapter 38

GRUB



38.1	The Grand Unified Boot Loader (GRUB)	478
38.2	Interactive Selections with GRUB at Boot	480
38.3	Installing GRUB	481
38.4	Customizing the GRUB Configuration	483
38.5	Boot Loader Specification Configuration (BLSCFG)	484
38.6	Labs	486

38.1 The Grand Unified Boot Loader (GRUB)

The Grand Unified Boot Loader (GRUB)

- Allows boot selection
 - Can choose alternate operating systems
 - Can choose alternate **Linux** kernels
 - Can choose alternate **initramfs** images
- Allows argument passing
 - Can change an existing stanza in menu editing mode
 - Can issue boot commands interactively on **GRUB** command line
- Version 2 has replaced Version 1 on all major **Linux** distributions except for **RHEL 6**-based ones

Virtually all **x86**-based **Linux** systems (outside the embedded sphere) today use **GRUB** (**G**rand **U**nified **Bo^olt**l**oader) to handle the early phases of system startup. Other platforms may have other equivalents, such as **ELILO** used on **EFI** systems such as **IA64** (**I**tanium), and **Das U-BOOT** used on many embedded configurations.**

Some important features of **GRUB** are:

- Alternative operating systems can be chosen at boot time.
- Alternative kernels and/or initial ramdisks can be chosen at boot time for a given operating system.
- Boot parameters can be easily changed at boot time without having to edit configuration files, etc. in advance.

For the exclusive use of LFS301 corp class taught 06 to 09 December

GRUB Versions

- **GRUB 2**

- Used on all modern **Linux** distributions
- Configuration Files:

```
/boot/grub/grub.cfg  
or /boot/grub2/grub.cfg  
or /boot/efi/EFI/redhat/grub.cfg  
/boot/grub2/grubenv  
/etc/default/grub  
/etc/grub.d
```

- **GRUB 1**

- Only major distribution still using is **RHEL 6**
- Configuration Files:
`/boot/grub/grub.conf`
- Philosophy the same, details quite different

- We will concentrate on **GRUB 2**; version 1 is now legacy

While details are different between **GRUB 2** and the legacy Version 1, the basic philosophy is the same.

At boot a basic configuration file is read:

```
/boot/grub/grub.cfg
```

or

```
/boot/grub2/grub.cfg
```

or

```
/boot/efi/EFI/redhat/grub.cfg
```

This file is auto-generated by **update-grub** (or **grub2-mkconfig**) based on configuration files in the `/etc/grub.d` directory and on `/etc/default/grub` and should not be edited by hand. Usually these utilities are run from other distribution-supplied scripts used for updating or compiling **Linux** kernels.

Once again the file is better off not being edited by hand. However, if you do make changes, most of the time they will be preserved.

For the exclusive use of LFS301 corp class taught 06 to 09 December

38.2 Interactive Selections with GRUB at Boot

Interactive Selections with GRUB at Boot

- Upon boot enters interactive shell
- Doing nothing picks default operating system and kernel
- Use up and down arrows for alternate selection
- Type `e` to enter interactive editor to change boot choices and parameters
- Changes will not be preserved in configuration files.

Upon system boot, after the initial POST and BIOS stages **GRUB** will be entered and display a menu containing a list of bootable images either from one or more **Linux** distributions or operating systems. There may also be submenus with even more choices.

Using up and down arrows and the `Enter` key the user can select the right boot option, or can wait for a configurable time period before the default choice is entered.

However, you can do much more. After selecting an entry, you can type `e` for edit and then enter into an interactive shell. In this shell you can alter the **stanza** in the configuration file that describes that particular boot option. Usually you do this to alter the **kernel command line**; for example adding the word `single` at the end of the command line will cause the system to boot in **single use mode** in order to take corrective actions. Once the desired change is made you can hit the right key to make the system boot.

At the bottom of the screen you will see displayed information on the exact key strokes, so there is no need to memorize.

Note that any changes you make to the configuration are **not persistent** and will be lost on the next boot. So for permanent changes, you need to change the actual files on the machine using the right utilities.

It is also possible to enter a pure shell, rather than edit a particular stanza. One can run a number of different commands and even try to re-install or repair **GRUB**. If there are serious problems, like not being able to find a configuration file, **GRUB** reverts back to this command line mode and you may be able to rescue the system without resorting to rescue media.

38.3 Installing GRUB

Installing GRUB

- The **GRUB** boot loader should be installed at system installation
 - Can be reinstalled later at any time, with **grub2-install** or **grub-install** depending on distribution
 - Install can also mean:
 - Setting up the configuration files grub needs to operate
 - Installing the needed utilities
- ```
$ sudo grub2-install /dev/sda
```
- Note on EFI multi-boot systems, you may have to run **efibootmgr** as well; things can be more complex. See the **man** page.

The word **installing** can have several different meanings with respect to **GRUB**:

1. Installing the **grub** program and associated utilities in their proper locations. In **GRUB 1** there was actually a program just called **grub**, but in **GRUB 2** there are a bunch of utilities with names like **grub2-\*** or **grub-\***.
2. Installing the files **GRUB** needs to operate at boot time, under either **/boot/grub** or **/boot/grub2**. This is separate than the files the **Linux** kernel needs (**vmlinuz-\***, **initramfs-\***) which will need to be in the **/boot** directory as well.
3. Installing **GRUB** as the **boot loader** in the system; usually this is done at the front of the entire hard disk, but can also be done in a partition.

To reinstall, the exact procedure for doing so depends on **GRUB** version. It can be as easy as:

```
$ sudo grub2-install /dev/sda
```

Please read the **man page** carefully before running such a command; there are many options, and messing up **GRUB** can make your system un-bootable.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# GRUB Device Nomenclature

- Needed to access files at boot time since file systems not mounted
- The first hard drive is denoted as `hd0`, the second as `hd1`, etc,
- However, partitions start counting from 1:
  - `sda1` is `(hd0,1)`
  - `sdc4` is `(hd2,4)`

Within the configuration file, each stanza has to specify what the `root` partition is; this is not the same as what we mean when we took about the root directory of the system. In this context it means the partition that contains the kernel itself (in the `/boot` directory.) For example it is very common to have `/boot` in its own partition, let us say `/dev/sda1`.

It is also fine to do `kernel (hd0,0)/vmlinuz ....` instead, and leave out the `root` line.

A quick look at `grub.cfg` should make it clear.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 38.4 Customizing the GRUB Configuration

### GRUB Configuration Files

- Do not edit `/boot/grub/grub.cfg` (or `/boot/grub2/grub.cfg`) directly
- Edit `/etc/default/grub` and the files in `/etc/grub.d`
- `grub.cfg` is rewritten every time `update-grub` is run.

```
c8:/tmp>cat /etc/default/grub
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR="$(sed 's, release .*$,,g' /etc/system-release)"
GRUB_DEFAULT=saved
GRUB_DISABLE_SUBMENU=true
GRUB_TERMINAL_OUTPUT="console"
GRUB_CMDLINE_LINUX="crashkernel=auto rhgb quiet"
GRUB_DISABLE_RECOVERY="true"
GRUB_ENABLE_BLSCFG=true
c8:/tmp>
```

Figure 38.1: `/etc/default/grub` on RHEL 8

- New file can be generated by `grub2-mkconfig` or `grub-mkconfig`

For **GRUB 2** two locations in the `/etc` directory require attention and are used to reconstruct `grub.cfg` whenever the system is altered with new kernels, or the relevant updating program (such as `update-grub` or `grubby`) is run manually.

The first one is `/etc/default/grub`. In addition, the directory `/etc/grub.d` matters. On **Ubuntu 19.10**:

```
student@ubuntu:~$ ls -l /etc/grub.d
total 80
-rwxr-xr-x 1 root root 9783 Mar 30 16:45 00_header
-rwxr-xr-x 1 root root 6258 Nov 1 2016 05_debian_theme
-rwxr-xr-x 1 root root 12676 Mar 30 16:45 10_linux
-rwxr-xr-x 1 root root 11281 Mar 30 16:45 20_linux_xen
-rwxr-xr-x 1 root root 1992 Jan 28 2016 20_memtest86+
-rwxr-xr-x 1 root root 12059 Mar 30 16:45 30_os-prober
-rwxr-xr-x 1 root root 1418 Mar 30 16:45 30_uefi-firmware
-rwxr-xr-x 1 root root 214 Mar 30 16:45 40_custom
-rwxr-xr-x 1 root root 216 Mar 30 16:45 41_custom
-rw-r--r-- 1 root root 483 Mar 30 16:45 README
student@ubuntu:~$
```

Figure 38.2: `/etc/grub.d` contents

Each of these files is run ascending order when the configuration file is updated.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 38.5 Boot Loader Specification Configuration (BLSCFG)

### BLSCFG

- **blscfg** (Boot Loader Specification Configuration) **grub** methods were introduced in **Fedora 30** and **RHEL 8**
- Transparent to normal user but where the **generated** configuration files are quite different:
  - `/boot/loader/entries`
  - `/boot/grub2/grubenv` (linked to `/boot/grub2/grub.cfg`)
- See [https://systemd.io/BOOT\\_LOADER\\_SPECIFICATION/](https://systemd.io/BOOT_LOADER_SPECIFICATION/)
- Can switch to with **grub2-switch-to-blscfg** if available
- Not clear if other **Linux** distributions will use

On systems configured with **BLSCFG** one still uses the usual **grub** commands when installing or updating kernels etc. When booting one will still see the same interactive **grub** screen and use it as it has always been done.

However, some differences are important. For example, while `/boot/grub2/grub.cfg` still exists, detailed information and options for each kernel that can be chosen does not appear there; it is in `/boot/loader/entries`, one file for each choice.

It is possible to switch to the new scheme by running the **grub2-switch-to-blscfg** program. One can also turn the new method on and off by altering the variable `GRUB_ENABLE_BLSCFG=[true|false]`. For example on a **RHEL 8** system:



`/etc/default/grub`

```
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR="$(sed 's, release .*$,,g' /etc/system-release)"
GRUB_DEFAULT=saved
GRUB_DISABLE_SUBMENU=true
GRUB_TERMINAL_OUTPUT="console"
GRUB_CMDLINE_LINUX="crashkernel=auto rhgb quiet"
GRUB_DISABLE_RECOVERY="true"
GRUB_ENABLE_BLSCFG=true
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

# /boot/loader/entries

```
$ sudo ls -l /boot/entries
```

```
-rw-r--r--. 1 root root 408 Jun 21 2019 ce0c82382a8a4c80bbd6931a917a2f1c-0-rescue.conf
-rw-r--r--. 1 root root 381 Feb 17 07:09 ce0c82382a8a4c80bbd6931a917a2f1c-4.18.0-240.15.1.el8_3.x86_64.conf
-rw-r--r--. 1 root root 285 Mar 9 06:41 ce0c82382a8a4c80bbd6931a917a2f1c-5.11.5.conf
-rw-r--r--. 1 root root 285 Mar 11 13:02 ce0c82382a8a4c80bbd6931a917a2f1c-5.11.6.conf
-rw-r--r--. 1 root root 305 Mar 15 07:23 ce0c82382a8a4c80bbd6931a917a2f1c-5.12.0-rc3.conf
```



## /boot/loader/entries/\*5.12.0-rc3.conf

```
title dracut (5.12.0-rc3) 8.2 (0otpa) dracut-049-70.git20200228.el8
version 5.12.0-rc3
linux /boot/vmlinuz-5.12.0-rc3
initrd /boot/initramfs-5.12.0-rc3.img $tuned_initrd
options $kernelopts $tuned_params
id dracut-20210315122247-5.12.0-rc3
grub_users $grub_users
grub_arg --unrestricted
grub_class kernel
```

With the boot loader specification configuration scheme, each possible kernel gains an entry in [/boot/loader/entries](#). A number of environmental variables may be set in [/boot/grub2/grubenv](#), including:

- `kernelopts`
- `tuned_initrd`
- `tuned_params`
- `grub_users`



## /boot/grub2/grubenv

```
GRUB Environment Block
kernelopts=root=UUID=6921b738-1e36-429a-89be-8b97cf2f0556 ro
boot_success=0
boot_ineterminate=0
saved_entry=ce0c82382a8a4c80bbd6931a917a2f1c-5.11.6.0~custom
#####
....
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 38.6 Labs



### Video Demonstration Resources

[using\\_grub\\_demo.mp4](#)

### Exercise 38.1: Booting into Non-Graphical Mode Using GRUB



#### Please Note

This exercise requires that it be run from the console (i.e., not over **SSH**).

1. Reboot your machine and go into the **GRUB** interactive shell by hitting **e** (or whatever other key is required as listed on your screen.)

2. Make your system boot into non-graphical mode. How you do this depends on the system.

On traditional systems that respect **runlevels** (which we will talk about in the next section) you can append a 3 to the kernel command line in the specific entry you pick from the **GRUB** menu of choices. This will still work on **systemd** systems that still bother to emulate **SysVinit** runlevels.

On some other systems you may need to append text instead.

3. Hit the proper key to make system continue booting.

4. After the system is fully operational in non-graphical mode, bring it up to graphical mode. Depending on your system, one of the following commands should do it:

```
$ sudo systemctl start gdm
$ sudo systemctl start lightdm
$ sudo telinit 5
$ sudo service gdm restart
$ sudo service lightdm restart
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 39

# System Init: systemd, SystemV and Upstart



|      |                      |     |
|------|----------------------|-----|
| 39.1 | The init Process     | 488 |
| 39.2 | Startup Alternatives | 489 |
| 39.3 | systemd              | 490 |
| 39.4 | systemctl            | 492 |
| 39.5 | Labs                 | 494 |

## 39.1 The init Process

# init

- `/sbin/init`
- *Parent of all processes*
- Coordinates rest of boot process
- Three most common implementations:
  - **systemd**
  - **Upstart**
  - **SysVinit**
- All major distributions have now moved to **systemd**

`/sbin/init` (usually just called **init**) is the first user-level process (or task) run on the system and continues to run until the system is shutdown. Traditionally it has been considered the **parent** of all user processes, although technically that is not true as some processes are started directly by the kernel.

**init** coordinates the later stages of the boot process, configures all aspects of the environment and starts the processes needed for logging into the system. **init** also works closely with the kernel in cleaning up after processes when they terminate.

Traditionally, nearly all distributions based the **init** process on **UNIX**'s venerable **SysVinit** software. However, this scheme was developed decades ago under rather different circumstances:

- The target was multi-user mainframe systems (and not personal computers, laptops, and other devices).
- The target was a single processor system.
- Startup (and shutdown) time was not an important matter; it was far less important than getting things right.

Startup was viewed as a **serial** process, divided into a series of sequential stages (termed **run levels**.) Each stage required completion before the next could proceed. Thus, startup did not easily take advantage of the parallel processing that could be done on multiple processors or cores.

Secondly, shutdown/reboot was seen as a relatively rare event and exactly how long it took was not considered important; today **Linux** systems usually boot in a manner of seconds.

Modern systems have required newer methods with enhanced capabilities.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 39.2 Startup Alternatives

# Startup Alternatives

- **Upstart**
  - Developed by **Ubuntu** and first included in 2006
  - Adopted in **Fedor**a 9 (in 2008) and in **RHEL 6** and its clones
  - Was also used in various embedded and mobile devices
- **systemd**
  - **Fedor**a was the first major distribution to adopt it in 2011
  - **RHEL** and **SUSE** followed
  - **Ubuntu 16.04** replaced **Upstart** with **systemd**.
  - All important **Linux** distributions are now based on **systemd**

To deal with the intrinsic limitations in **SysVinit**, new methods of controlling system startup were developed. While there are others, two main schemes were adopted by Enterprise distributors, **Upstart** and **systemd**.

Every other major **Linux** distribution has now adopted **systemd** and configured it as the default. Even **Ubuntu** phased out **Upstart** in its favor.

Migration to **systemd** was non-trivial and bugs and missing features could be very disabling, so essential compatibility layers were adopted and still exist for legacy software. Thus, **SysVinit** utility compatibility wrappers still persist.

The history of **systemd** development and adoption is rather complicated, and colorful personalities ensured not all the discussion was both friendly and technical. But this holy war is over as far as we are concerned here.

In the following we will concentrate on **systemd**, and, for the most part, ignore **SysVinit** as well as **Upstart**, which is no longer used in any significant way.

For the exclusive use of LFS301 corp class taught 06 to 09 December

### 39.3 systemd

## systemd Features

- Boots faster than previous init systems
- Provides aggressive parallelization capabilities
- Uses **socket** and **D-Bus** activation for starting services
- Replaces shell scripts with programs
- Offers on-demand starting of daemons
- Keeps track of processes using **cgroups**
- Maintains mount and automount points
- Implements an elaborate transactional dependency-based service control logic
- Can work as a drop-in replacement for **SysVinit** and is compatible with **SysVinit** scripts.

The **systemd** system and session manager for **Linux** is now dominant in all major distributions.

Note that **systemd** is backward compatible with **SysVinit** and the concept of runlevels is supported via runlevel **targets**. The **telinit** program is emulated to work with runlevels.

Instead of **bash** scripts, **systemd** uses **.service** files. In addition, **systemd** sorts all daemons into their own **Linux cgroups** (control groups).

For the exclusive use of LFS301 corp class taught 06 to 09 December

# systemd Configuration Files

- `/etc/hostname` replaced:
  - `/etc/sysconfig/network` in **Red Hat**-based systems
  - `/etc/HOSTNAME` in **SUSE**-based systems
  - `/etc/hostname` (was already adopted as the standard) in **Debian**-based systems.
- Other files might include:
  - `/etc/vconsole.conf`: default keyboard mapping and console font
  - `/etc/sysctl.d/*.conf`: drop-in directory for kernel **sysctl** parameters
  - `/etc/os-release`: distribution ID file

Although **systemd** prefers to use a set of standardized configuration files, it can use distribution-dependent legacy configuration files as a fall-back.

Exactly which configuration files will depend on how each distribution sets things up; for example, `/etc/vconsole.conf`, which configures virtual terminal defaults, does not appear on **Ubuntu** systems.



## Compatibility with SysVinit

**systemd** is configured to be mostly backward compatible with **SysVinit** so using old commands will generally work.

It supports the use of runlevels conceptually, through the mechanism of runlevel *targets*. In addition, **telinit** is emulated to work with runlevels.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 39.4 systemctl

# systemctl

```
$ systemctl [options] command [name]
```

- To show the status of everything **systemd** controls:

```
$ systemctl
```

- Show all available services:

```
$ systemctl list-units -t service --all
```

- Show only active services:

```
$ systemctl list-units -t service
```

- To start (activate) one or more **units**:

```
$ sudo systemctl start foo
```

```
$ sudo systemctl start foo.service
```

```
$ sudo systemctl start /path/to/foo.service
```

where a unit can be a service or a socket.

**systemctl** is the main utility for managing services. Some **systemctl** commands can be run as non-root user, others require running as root or with **sudo**.

For most commands you can omit the `.service` attached to the service name.

For an excellent summary of how to go from **SysVinit** to **systemd**, see the **SysVinit to systemd Cheatsheet** at [https://fedoraproject.org/wiki/SysVinit\\_to\\_Systemd\\_Cheatsheet](https://fedoraproject.org/wiki/SysVinit_to_Systemd_Cheatsheet).

## systemctl (cont'd)

- To stop (deactivate):

```
$ sudo systemctl stop foo.service
```

- Enable/disable a service:

```
$ sudo systemctl enable sshd.service
$ sudo systemctl disable sshd.service
```

These commands do not actually start or stop a service; they control whether or not it is started up at system boot.

For most commands you can omit the .service attached to the service name.

## 39.5 Labs



### Video Demonstration Resources

[using\\_systemctl\\_demo.mp4](#)

### Exercise 39.1: Adding a New Startup Service with systemd

To add a new startup service we need to create (as root) a file directly under `/etc/systemd/system` or somewhere else in that directory tree; distributions have some varying tastes on this. For example a very minimal file named `/etc/systemd/system/fake2.service` (which can be extracted from your downloaded SOLUTIONS file as `fake2.service`) containing the following:

#### fake2.service

```
[Unit]
Description=fake2
After=network.target

[Service]
ExecStart=/bin/sh -c '/bin/echo I am starting the fake2 service ; /bin/sleep 30'
ExecStop=/bin/echo I am stopping the fake2 service

[Install]
WantedBy=multi-user.target
```

Now there are many things that can go in this **unit** file. The `After=network.target` means the service should start only after the network does, while the `WantedBy=multi-user.target` means it should start when we reach multiple-user mode. This is equivalent to runlevels 2 and 3 in **SysVinit**. Note `graphical.target` would correlate with runlevel 5.

Now all we have to do to start, stop and check the service status are to issue the commands:

```
$ sudo systemctl start fake2.service
$ sudo systemctl status fake2.service
$ sudo systemctl stop fake2.service
```

If you are fiddling with the unit file while doing this you'll need to reload things with:

```
$ sudo systemctl daemon-reload
```

as the system will warn you.

To keep an eye directly on the output you can do:

```
$ sudo tail -f /var/log/messages
```

(use `/var/log/syslog` on **Ubuntu**) either in background or in another windows while the service is running.

To set things up so the service turns on or off on system boot:

```
$ sudo systemctl enable fake2.service
$ sudo systemctl disable fake2.service
```

Once again, you really need to reboot to make sure it has taken effect.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 40

# Backup and Recovery Methods



|       |                                                       |     |
|-------|-------------------------------------------------------|-----|
| 40.1  | Backup Basics . . . . .                               | 496 |
| 40.2  | Backup vs Archive . . . . .                           | 498 |
| 40.3  | Backup Methods and Strategies . . . . .               | 500 |
| 40.4  | tar . . . . .                                         | 503 |
| 40.5  | Compression: gzip, bzip2 and xz and Backups . . . . . | 506 |
| 40.6  | dd . . . . .                                          | 507 |
| 40.7  | rsync . . . . .                                       | 508 |
| 40.8  | cpio ** . . . . .                                     | 509 |
| 40.9  | Backup Programs ** . . . . .                          | 510 |
| 40.10 | Labs . . . . .                                        | 511 |



### Please Note

\*\* These sections may be considered in part or in whole as optional. They contain either background reference material, specialized topics, or advanced subjects. The instructor may choose to cover or not cover them depending on classroom experience and time constraints.

## 40.1 Backup Basics

### Why Backups?

- Data is valuable
- Hardware fails
- Software fails
- People make mistakes
- Malicious people can cause deliberate damage
- Unexplained events happen
- Rewinds can be useful

Whether you are administering only one personal system or a network of many machines, system backups are very important. On-disk data is an important work product and is therefore a commodity that we wish to protect. Re-creating lost data costs time and money. Some data may even be unique and there may be no way to re-create it.

While storage media reliability has increased, so has drive capacity. Even if the failure rate per byte decreases, unpredictable failures still occur. It may be pessimistic to say there are only two kinds of drives; those that have failed and those that will fail, but it is essentially true. Using **RAID** helps but backups are still needed.

No software is perfect. Bugs may destroy or corrupt data. Even stable programs long in use can have problems.

Everyone has heard **OOPS!** (or something much worse) coming from the next cubicle (or from their own mouth) at one time or another. Sometimes just a simple typing error can cause large scale destruction of files and data.

It could be the canonical disgruntled employee or an external hacker with a point to make. Security concerns and backup capabilities are very strongly related.

Files can just disappear without you knowing how, who, or even when it occurred.

Sometimes restoring to an earlier **snapshot** of all or part of the system may be required.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# What Do We Need to Backup?

- Definitely:
  - Business-related data.
  - System configuration files.
  - User files (usually under `/home`).
- Maybe:
  - Spooling directories (for printing, mail etc.)
  - Logging files (found in `/var/log`, and elsewhere).
- Probably not:
  - Software that can easily be re-installed; on a well-managed system this should be almost everything.
  - The `/tmp` directory
- Definitely not:
  - Pseudo-filesystems such as `/proc`, `/dev` and `/sys`, and swap

Obviously, files essential to your organization require backup. Configuration files may change frequently, and along with individual user's files, require backup as well.

Logging files can be important if you have to investigate your system's history, which can be particularly important for detecting intrusions and other security violations.

You do not have to back up anything that can easily be re-installed. Also the swap partitions (or files) and `/proc` filesystems are generally not useful or necessary to backup since data in these areas is basically temporary (just like in the `/tmp` directory).

## 40.2 Backup vs Archive

### Backup vs Archive

- All backup media have a finite life time before becoming unreadable
- Conventional Estimates:
  - Magnetic Tapes: 10-30 years
  - CDs and /DVDs: 3-10 years
  - Hard Disks: 2-5 years
- Lifetime very sensitive to:
  - Environmental conditions (temperature, humidity etc.)
  - Quality of media
  - Having working software that can read data on current operating systems and hardware
- Lifetime is sufficient for backup; not for permanent digital archiving

For life times longer than the usual back up timescale, data can be preserved using multiple copies plus copying over to newer media from time to time.

For very long times (i.e., many decades, centuries etc.) standard methods do not work easily as everything can go obsolete:

- Hardware
- Software and Document Format
- Media

None of the inexpensive digital formats can actually compete with paper and film for long periods (if they are properly stored and continuously cared for – like wine.)

This is a problem serious people think about and there should be good solutions available before all is lost.

# Tape Drives

- Relatively slow
- Permit only sequential access
- Not as common as they once were
- Rarely used for primary backup today
- Still sometimes useful for off-site storage and archiving

Tape drives are not as common as they used to be. They are relatively slow and permit only sequential access. On any modern setup they are rarely used for primary backup. They are sometimes used for off-site storage for archival purposes for long time references. However, magnetic tape drives always have only a finite lifetime without physical degradation and loss of data.

Modern tape drives are usually of the **LTO** (Linear Tape Open) variety, whose first versions appeared in the late 1990s as an open standards alternative; early formats were mostly proprietary. Early versions held up to 100 GB; newer versions can hold 2.5 TB or more in a cartridge of the same size.

Day to day backups are usually done with some form of **NAS** (Network Attached Storage) or with **cloud**-based solutions, making new tape-based installations less and less attractive. However, they can still be found and system administrators may be required to deal with them.

In what follows we will try not to focus on particular physical forms for the backup media, and will speak more abstractly.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 40.3 Backup Methods and Strategies

### Backup Methods

- **Full:**

Backup all files on the system.

- **Incremental:**

Backup all files that have changed since the last incremental or full backup.

- **Differential:**

Backup all files that have changed since the last full backup.

- **Multiple level incremental:**

Backup all files that have changed since the previous backup at the same or a previous level.

- **User:**

Backup only files in a specific user's directory.

Several different kinds of backup methods can be used, often in concert with each other as listed.

One should never have all backups residing in the same physical location as the systems being protected. Otherwise, fire or other physical damage could lead to a total loss. In the past this usually meant physically transporting magnetic tapes to a secure location. Today this is more likely to mean transferring backup files over the Internet to alternative physical locations. Obviously, this has to be done in a secure way using encryption and other security precautions as is appropriate.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Backup Strategies

One useful strategy involving tapes (you can easily substitute other media in the description):

1. Use tape 1 for a full backup on Friday.
2. Use tapes 2-5 for incremental backups on Monday-Thursday.
3. Use tape 6 for full backup on second Friday.
4. Use tapes 2-5 for incremental backups on second Monday-Thursday.
5. Do not overwrite tape 1 until completion of full backup on tape 6.
6. After full backup to tape 6, move tape 1 to external location for disaster recovery.
7. For next full backup (next Friday) exchange tape 1 for tape 6.

We should note that backup methods are useless without associated **restore** methods. One has to take into account the robustness, clarity and ease of both directions when selecting strategies.

A good rule of thumb is to have at least two weeks of backups available.

The simplest backup scheme is to do a full backup of everything once, and then perform incremental backups of everything that subsequently changes. While full backups can take a lot of time, restoring from incremental backups can be more difficult and time consuming. Thus, one can use a mix of both to optimize time and effort.

# Some Backup Related Utilities

- **cpio**
- **tar**
- **gzip, bzip2, xz**
- **dd**
- **rsync**
- **dump/restore**
- **mt**

**cpio** and **tar** create and extract **archives** of files.

The archives are often compressed with **gzip**, **bzip2**, or **xz**. The archive file may be written to disk, magnetic tape, or any other device which can hold files. Archives are very useful for transferring files from one filesystem or machine to another.

**dd** is a powerful utility often used to transfer raw data between media. It can be used to copy entire partitions or entire disks.

**rsync** is a powerful utility that can synchronize directory subtrees or entire filesystems across a network, or between different filesystem locations on a local machine.

**dump** and **restore** are ancient utilities which were designed specifically for backups. They read from the filesystem directly (which is more efficient). However, they must be restored only on the same filesystem type that they came from. There are newer alternatives.

**mt** is used for querying and positioning tapes before performing backups and restores.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 40.4 tar

### Using tar for Backups

- Create an archive: use `-c` or just `c`

```
$ tar cvf /dev/st0 /root
$ tar -cvf /dev/st0 /root
```

- Create with multi volume option: use `-M`

```
$ tar -cMf /dev/st0 /root
```

Will be prompted to put next tape in

- Verify files with compare option: use `-d` or `--compare`

```
$ tar --compare --verbose --file /dev/st0
$ tar -dvf /dev/st0
```

- Note each option has a short form (one letter with a `-`) or a long form (with `--`)

- **tar** is by default recursive

When creating a **tar** archive, for each directory given as an argument, all files and subdirectories will be included in the archive. When restoring it reconstitutes directories as necessary.

It even has a `--newer` option that lets you do incremental backups.

The version of **tar** used in **Linux** can also handle backups that do not fit on one tape or whatever device you use.

You can specify a device or file with the `-f` or `--file` options.

After you make a backup, you can make sure that it is complete and correct using the verification option.

By default **tar** will recursively include all subdirectories in the archive.

When you create an archive, **tar** prints a message about removing leading slashes from the absolute path name. While this allows you to restore the files anywhere, the default behavior can be modified.

Most **tar** options can be given in short form with one dash, or long form with two: `-c` is completely equivalent to `--create`.

Also note that you can combine options (when using the short notation) so that you do not have to type every dash.

Furthermore, single-dashed **tar** options can be used with or without dashes; i.e., `tar cvf file.tar dir1` has the same result as `tar -cvf file.tar dir1`.

# Using tar for Restoring Files

- Extract from an archive: use -x or --extract

```
$ tar --extract --same-permissions --verbose --file /dev/st0
$ tar -xpvf /dev/st0
$ tar xpvf /dev/st0
```

- You may name specific files to restore

```
$ tar -xvf /dev/st0 somefile
```

- Listing the contents of a tar backup

```
$ tar --list --file /dev/st0
$ tar -tf /dev/st0
```

The -x or --extract option extracts files from an archive, all by default. One can narrow the file extraction list by specifying only particular files. If a directory is specified, all included files and subdirectories are also extracted.

The -p or --same-permissions option ensures files are restored with their original permissions.

The -t or --list option lists, but does not extract, the files in the archive.

## Incremental Backups with tar

- Use `--newer` or `--after-date` option
- Based on date
- Create a backup of all files modified after a certain date and compress them:

```
$ tar --create --newer '2011-12-1' -vzf backup1.tgz /var/tmp
$ tar --create --after-date '2011-12-1' -vzf backup1.tgz /var/tmp
```

Either form creates a backup archive of all files in `/var/tmp` which were modified after December 1, 2011.

You can do an incremental backup with `tar` using the `-N` (or the equivalent `--newer`), or the `--after-date` options. Either option requires specifying either a date or a qualified (reference) file name.

Because `tar` only looks at a file's date, it does not consider any other changes to the file, such as permissions or file name. To include files with these changes in the incremental backup, use `find` and create a list of files to be backed up.

**Note:** When followed by an option like `--newer` you must use the dash in options like `-vzf`, or `tar` will get confused. This kind of option specification confusion sometimes occurs with old **UNIX** utilities like `ps` and `tar` with complicated histories involving different families of **UNIX**.

## 40.5 Compression: gzip, bzip2 and xz and Backups

### Archive Compression Methods

- **Compressing** files saves disk space and/or network transmission time.
- In order of increasing compression efficiency which comes at the cost of longer compression times:
  - **gzip**: Uses Lempel-Ziv Coding (LZ77), and produces .gz files
  - **bzip2**: Uses Burrows-Wheeler block sorting text compression algorithm and Huffman coding, and produces .bz2 files
  - **xz**: Produces .xz files. and also supports legacy .lzma format
- Modern machines will often find the **compress** → **transmit** → **decompress** cycle faster than just transmitting (or copying) an uncompressed file.
- <https://www.kernel.org> only uses **xz** format now for downloading **Linux** kernels

The compression utilities are very easily (and often) used in combination with **tar**:

```
$ tar zcvf source.tar.gz source
$ tar jcvf source.tar.bz2 source
$ tar Jcvf source.tar.xz source
```

for producing a compressed archive. Note the first command has the exact same effect as:

```
$ tar cvf source.tar source ; gzip -v source.tar
```

but is more efficient because there is no intermediate file storage, and archiving and compression happen simultaneously in the pipeline.

For decompression:

```
$ tar xzvf source.tar.gz
$ tar xjvf source.tar.bz2
$ tar xJvf source.tar.xz
```

or even simpler:

```
$ tar xvf source.tar.gz
```

as modern versions of **tar** can sense the method of compression and take care of it automatically.

Obviously it is not worth using these methods on archives whose component files are already compressed, such as images, or .pdf files etc.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 40.6 dd

# dd

- Used to copy raw data
- Low level copy of data
- Syntax: dd if=input-file of=output-file options  
    \$ dd if=/dev/sda of=/dev/sdb
- Can be used to create files  
    \$ dd if=/dev/zero of=file1 bs=1M count=10
- Can backup entire hard drives or partitions
- Can copy CD or DVDs

**dd** is a common **UNIX**-based program whose primary purpose is the low-level copying and conversion of raw data. It is used to copy a specified number of bytes or blocks, performing on-the-fly byte order conversions, as well as being able to convert data from one form to another. It can also be used to copy regions of raw device files, for example backing up the boot sector of a hard disk, or to read fixed amounts of data from special files like `/dev/zero` or `/dev/random`.

Backup an entire hard drive to another.

```
$ dd if=/dev/sda of=/dev/sdb
```

Create an image of a hard disk.

```
$ dd if=/dev/sda of=sdadisk.img
```

Backup a partition.

```
$ dd if=/dev/sda1 of=partition1.img
```

Backup a CD ROM.

```
$ dd if=/dev/cdrom of=tgsservice.iso bs=2048
```

## 40.7 rsync

# Using rsync for Backups

- Usage:

```
$ rsync [options] sourcefile destinationfile
```

- Between local machine and a network target:

```
$ rsync file.tar someone@backup.mydomain:/usr/local
```

- Test before doing with --dry-run:

```
$ rsync -r --dry-run /usr/local /BACKUP/usr
```

**rsync** (remote synchronize) is used to transfer files across a network (or between different locations on the same machine).

The source and destination can take the form of target:path where target can be in the form of [user@]host. The user@ part is optional and used if the remote user is different from the local user. Thus, these are all possible **rsync** commands:

```
$ rsync file.tar someone@backup.mydomain:/usr/local
```

```
$ rsync -r --dry-run /usr/local /BACKUP/usr
```

You have be very careful with **rsync** about exact location specifications (especially if you use the --delete option), so it is highly recommended to use the --dry-run option first, and then repeat the command if the projected action looks correct.

**rsync** is very clever; it checks local files against remote files in small chunks, and it is very efficient in that when copying one directory to a similar directory, only the differences are copied over the network with the first directory. One often uses the -r option which causes **rsync** to recursively walk down the directory tree copying all files and directories below the one listed as the sourcefile. Thus, a very useful way to back up a project directory might be similar to:

```
$ rsync -r project-X archive-machine:archives/project-X
```

A simple (and very effective and very fast) backup strategy is to simply duplicate directories or partitions across a network with **rsync** commands and to do so frequently.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 40.8 cpio \*\*

# Using cpio for Backups

- Create an archive: use -o or --create  
`$ ls | cpio --create -O /dev/st0`
- Extract from an archive: use -i or --extract  
`$ cpio -i -I /dev/st0`
- List contents of an archive: use -t or --list  
`$ cpio -t -I /dev/st0`

**cpio** (**c**opy **i**n and **o**ut) is a general file archiver utility that has been around since the earliest days of **UNIX** and was originally designed for tape backups. Even though newer archiving programs (like **tar** which is not exactly young) have been deployed to do many of the tasks that were once in the domain of **cpio**, it still survives.

The **rpm2cpio** utility can be used to convert **RPM** packages into **cpio** archives and then extract them. Also the **Linux** kernel uses a version of **cpio** internally to deal with **initramfs** and **initrd** initial ram filesystems and disks during boot. One reason **cpio** lives on is that it is lighter than **tar** and other successors, even if it is somewhat less robust.

You can specify the input (-I device) or output (-O device) or use redirection on the command line.

The -o or --create option tells **cpio** to copy files out to an archive. **cpio** reads a list of file names (one per line) from standard input and writes the archive to standard output.

The -i or --extract option tells **cpio** to copy files in from an archive, reading the archive from standard input. If you list file names as patterns (such as \*.c) on the command line, only files in the archive that match the patterns are copied from the archive. If no patterns are given, all files are extracted.

The -t or --list option tells **cpio** to list the archive contents. Adding the -v or --verbose option generates a long listing.

## 40.9 Backup Programs \*\*

# Backup Programs

- **Amanda**
- **Bacula**
- **Clonezilla**

There is no shortage of available backup program suites available for **Linux**, including proprietary applications or those supplied by storage vendors, as well as open-source applications. Several that are particularly well known are:

- **Amanda** (**A**dvanced **M**aryland **A**utomatic **N**etwork **D**isk **A**rchiver) uses native utilities (including **tar** and **dump**) but is far more robust and controllable.  
**Amanda** is generally available on Enterprise **Linux** systems through the usual repositories. Complete information can be found at <http://www.amanda.org>.
- **Bacula** is designed for automatic backup on heterogeneous networks. It can be rather complicated to use and is recommended (by its authors) only to experienced administrators.  
**Bacula** is generally available on Enterprise **Linux** systems through the usual repositories. Complete information can be found at [https://www.bacula.org/7.0.x-manuals/en/main/Main\\_Reference.html](https://www.bacula.org/7.0.x-manuals/en/main/Main_Reference.html).
- **Clonezilla** is a very robust **disk cloning** program, which can make images of disks and deploy them, either to restore a backup, or to be used for **ghosting**, to provide an image that can be used to install many machines. Complete information can be found at <https://clonezilla.org>.

The program comes in two versions: **Clonezilla Live**, which is good for single machine backup and recovery, and **Clonezilla SE, server edition**, which can clone to many computers at the same time.

**Clonezilla** is not very hard to use and is extremely flexible, supporting many operating systems (not just **Linux**), file system types, and boot loaders.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 40.10 Labs

### Exercise 40.1: Using tar for Backup

1. Create a directory called `backup` and in it place a compressed `tar` archive of all the files under `/usr/include`, with the highest level directory being `include`. You can use any compression method (`gzip`, `bzip2` or `xzip`).
2. List the files in the archive.
3. Create a directory called `restore` and unpack and decompress the archive.
4. Compare the contents with the original directory the archive was made from.

### Solution 40.1

```
1. $ mkdir /tmp/backup
$ cd /usr ; tar zcvf /tmp/backup/include.tar.gz include
$ cd /usr ; tar jcvf /tmp/backup/include.tar.bz2 include
$ cd /usr ; tar Jcvf /tmp/backup/include.tar.xz include
```

or

```
$ tar -C /usr -zcf include.tar.gz include
$ tar -C /usr -jcf include.tar.bz2 include
$ tar -C /usr -Jcf include.tar.xz include
```

Notice the efficacy of the compression between the three methods:

```
$ du -sh /usr/include
```

|   |     |              |
|---|-----|--------------|
| 1 | 55M | /usr/include |
|---|-----|--------------|

2. \$ ls -lh include.tar.\*

|   |                                                          |
|---|----------------------------------------------------------|
| 1 | c8:/tmp/backup>ls -lh                                    |
| 2 | total 17M                                                |
| 3 | -rw-rw-r-- 1 coop coop 5.3M Jul 18 08:17 include.tar.bz2 |
| 4 | -rw-rw-r-- 1 coop coop 6.7M Jul 18 08:16 include.tar.gz  |
| 5 | -rw-rw-r-- 1 coop coop 4.5M Jul 18 08:18 include.tar.xz  |

3. \$ tar tvf include.tar.xz

|   |                                                              |
|---|--------------------------------------------------------------|
| 1 | qdrwxr-xr-x root/root 0 2014-10-29 07:04 include/            |
| 2 | -rw-r--r-- root/root 42780 2014-08-26 12:24 include/unistd.h |
| 3 | -rw-r--r-- root/root 957 2014-08-26 12:24 include/re_comp.h  |
| 4 | -rw-r--r-- root/root 22096 2014-08-26 12:24 include/regex.h  |
| 5 | -rw-r--r-- root/root 7154 2014-08-26 12:25 include/link.h    |
| 6 | .....                                                        |

Note it is not necessary to give the `j`, `J`, or `z` option when decompressing; `tar` is smart enough to figure out what is needed.

4. \$ cd .. ; mkdir restore ; cd restore  
\$ tar xvf ../backup/include.tar.bz2

|   |                                  |
|---|----------------------------------|
| 1 | include/                         |
| 2 | include/unistd.h                 |
| 3 | include/re_comp.h                |
| 4 | include/regex.h                  |
| 5 | include/link                     |
| 6 | .....                            |
| 7 | \$ diff -qr include /usr/include |

**For the exclusive use of LFS301 corp class taught 06 to 09 December**

## Exercise 40.2: Using cpio for Backup

We are going to do essentially the same exercise now, but using **cpio** in place of **tar**. We'll repeat the slightly altered instructions for ease of use.

1. Create a directory called `backup` and in it place a compressed **cpio** archive of all the files under `/usr/include`, with the highest level directory being `include`. You can use any compression method (**gzip**, **bzip2** or **xzip**).
2. List the files in the archive.
3. Create a directory called `restore` and unpack and decompress the archive.
4. Compare the contents with the original directory the archive was made from.

## Solution 40.2

1. `$ (cd /usr ; find include | cpio -c -o > /home/student/backup/include.cpio)`

```
1 82318 blocks
```

or to put it in a compressed form:

`$ (cd /usr ; find include | cpio -c -o | gzip -c > /home/student/backup/include.cpio.gz)`

```
1 82318 blocks
```

`$ ls -lh include*`

```
1 total 64M
2 -rw-rw-r-- 1 coop coop 41M Nov 3 15:26 include.cpio
3 -rw-rw-r-- 1 coop coop 6.7M Nov 3 15:28 include.cpio.gz
4 -rw-rw-r-- 1 coop coop 5.3M Nov 3 14:44 include.tar.bz2
5 -rw-rw-r-- 1 coop coop 6.8M Nov 3 14:44 include.tar.gz
6 -rw-rw-r-- 1 coop coop 4.7M Nov 3 14:46 include.tar.xz
```

2. `$ cpio -ivt < include.cpio`

```
1 drwxr-xr-x 86 root root 0 Oct 29 07:04 include
2 -rw-r--r-- 1 root root 42780 Aug 26 12:24 include/unistd.h
3 -rw-r--r-- 1 root root 957 Aug 26 12:24 include/re_comp.h
4 -rw-r--r-- 1 root root 22096 Aug 26 12:24 include/regex.h
5
```

Note the redirection of input; the archive is not an argument. One could also do:

```
$ cd ../restore
$ cat ../backup/include.cpio | cpio -ivt
$ gunzip -c include.cpio.gz | cpio -ivt
```

3. `$ rm -rf include`  
`$ cpio -id < ../backup/include.cpio`  
`$ ls -lR include`

or

```
$ cpio -idv < ../backup/include.cpio
$ diff -qr include /usr/include
```

## Exercise 40.3: Using rsync for Backup

For the exclusive use of LFS301 corp class taught 06 to 09 December

1. Using **rsync**, we will again create a complete copy of `/usr/include` in your backup directory:

```
$ rm -rf include
$ rsync -av /usr/include .

1 sending incremental file list
2 include/
3 include/FlexLexer.h
4 include/_G_config.h
5 include/a.out.h
6 include/aio.h
7
```

2. Let's run the command a second time and see if it does anything:

```
$ rsync -av /usr/include .

1 sending incremental file list
2
3 sent 127398 bytes received 188 bytes 255172.00 bytes/sec
4 total size is 41239979 speedup is 323.23
```

3. One confusing thing about **rsync** is you might have expected the right command to be:

```
$ rsync -av /usr/include include
1 sending incremental file list
2 ...
```

However, if you do this, you'll find it actually creates a new directory, `include/include`!

4. To get rid of the extra files you can use the `--delete` option:

```
$ rsync -av --delete /usr/include .
1 sending incremental file list
2 include/
3 deleting include/include/xen/privcmd.h
4 deleting include/include/xen/evtchn.h
5
6 deleting include/include/FlexLexer.h
7 deleting include/include/
8
9 sent 127401 bytes received 191 bytes 85061.33 bytes/sec
10 total size is 41239979 speedup is 323.22
```

5. For another simple exercise, remove a subdirectory tree in your backup copy and then run **rsync** again with and without the `--dry-run` option:

```
$ rm -rf include/xen
$ rsync -av --delete --dry-run /usr/include .

1 sending incremental file list
2 include/
3 include/xen/
4 include/xen/evtchn.h
5 include/xen/privcmd.h
6
7 sent 127412 bytes received 202 bytes 255228.00 bytes/sec
8 total size is 41239979 speedup is 323.16 (DRY RUN)
```

```
$ rsync -av --delete /usr/include .
```

6. A simple script with a good set of options for using **rsync**:

For the exclusive use of LFS301 corp class taught 06 to 09 December



## script using rsync

```
#!/bin/sh
set -x

rsync --progress -avrxH --delete $*
```

which will work on a local machine as well as over the network. Note the important `-x` option which stops `rsync` from crossing filesystem boundaries.

### Extra Credit

For more fun, if you have access to more than one computer, try doing these steps with source and destination on different machines.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 41

# Linux Security Modules



|      |                                  |     |
|------|----------------------------------|-----|
| 41.1 | Linux Security Modules . . . . . | 516 |
| 41.2 | SELinux . . . . .                | 518 |
| 41.3 | AppArmor . . . . .               | 530 |
| 41.4 | Labs . . . . .                   | 534 |

## 41.1 Linux Security Modules

# What are Linux Security Modules?

- An **LSM** (Linux Security Module) implements **mandatory access controls** on requests made to the kernel.
- Use of an **LSM**:
  - Minimizes changes to the kernel.
  - Minimizes overhead on the kernel.
  - Permits flexibility and choice between different implementations, each of which is presented as a self-contained **LSM** (Linux Security Module).

A modern computer system must be made secure, but needs vary according to sensitivity of data, number of users with accounts, exposure to outside networks, legal requirements and other factors. Responsibility for enabling good security controls falls both on application designers and the **Linux** kernel developers and maintainers. Of course users have to follow good procedures as well, but on a well run system, non-privileged users should have very limited ability to expose the system to security violations. In this section we are concerned with how the **Linux** kernel enhances security through the use of the **Linux Security Modules** framework, particularly with the deployment of **SELinux**.

The basic idea is to **hook system calls**; insert code whenever an application requests a transition to kernel (system) mode in order to accomplish work that requires enhanced abilities; this code makes sure permissions are valid, malicious intent is protected against, etc. It does this by invoking security-related functional steps before and/or after a system call is fulfilled by the kernel.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Main LSM Choices

- **SELinux**: [https://selinuxproject.org/page/Main\\_Page](https://selinuxproject.org/page/Main_Page)
- **AppArmor**: <https://apparmor.net>
- **Smack**: <http://schaufler-ca.com>
- **TOMOYO**: <https://tomoyo.osdn.jp>



## Use of Simultaneous LSMs

Since 2019 it has been possible to combine (stack) **LSMs** in certain specified orders.

- See <https://www.starlab.io/blog/a-brief-tour-of-linux-security-modules> for a nice review of the mechanism and choices.

For a long time the only enhanced security model implemented was **SELinux**. When the project was first floated upstream in 2001 to be included directly in the kernel, there were objections about using only one approach to enhanced security.

As a result the **LSM** approach was adopted where alternative modules to **SELinux** could be used as they were developed and was incorporated into the **Linux** kernel in 2003.

Originally, only one **LSM** could be used at a time as they can potentially modify the same parts of the **Linux** kernel.

However, it is now possible to stack them with some care. **LSMs** are now considered as either **major** or **minor** when configuring their combination.

We will concentrate primarily on **SELinux** and secondarily on **AppArmor** in order of usage volume.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 41.2 SELinux

# SELinux Overview

- Developed by the NSA
- Additional method of protection
- Uses:
  - Contexts
  - Rules
  - Policies
- Actions not explicitly allowed are denied, by default
- Additional Online Resources:

### **SELinux User's and Administrator's Guide:**

[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/7/html/selinux\\_users\\_and\\_administrators\\_guide/index](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/selinux_users_and_administrators_guide/index)

**SELinux** was originally developed by the United States **NSA** (National Security Administration) and has been integral to **RHEL** for a very long time, which has brought it a large usage base.

Operationally, **SELinux** is a set of security rules that are used to determine which processes can access which files, directories, ports, and other items on the system.

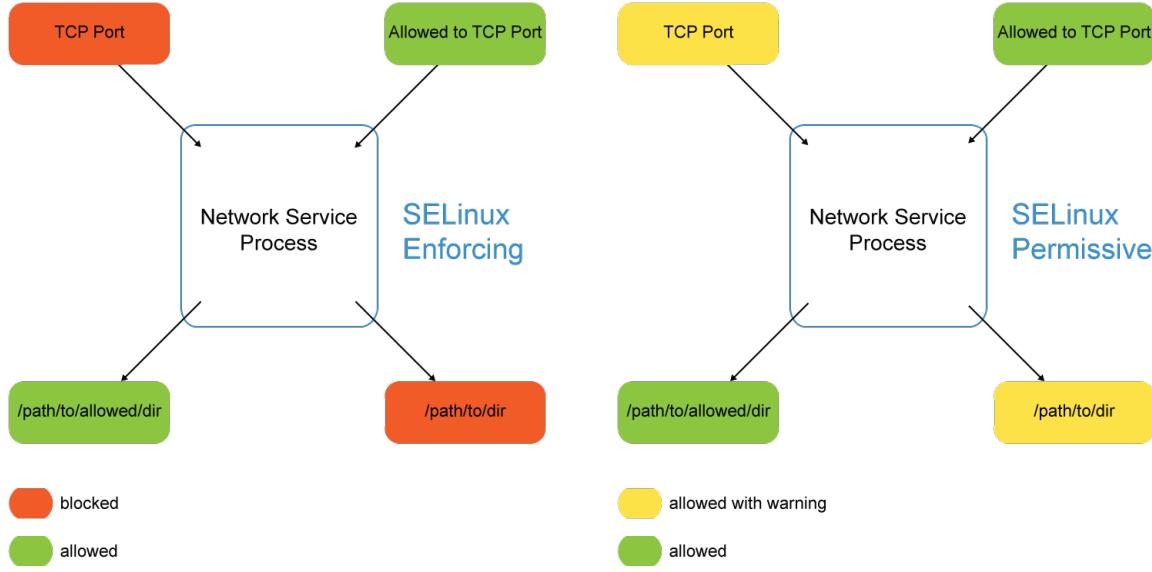
It works with these three conceptual quantities:

1. **Contexts:** Are labels to files, processes and ports. Examples of contexts are **SELinux** user, role and type.
2. **Rules:** Describe access control in terms of **contexts**, **processes**, **files**, **ports**, **users**, etc.
3. **Policies:** Are a set of **rules** that describes what system-wide access control decisions should be made by **SELinux**.

An **SELinux** context is a label used by a rule to define how users, processes, files and ports interact with each other. As the default policy is to deny any access, rules are used to describe allowed actions on the system.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## SELinux Enforcement Modes I



These modes are selected (and explained) in a file (usually `/etc/selinux/config`) whose location varies by distribution (it is often either at `/etc/sysconfig/selinux` or linked from there). The file is well self-documented.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# SELinux Enforcement Modes II

- Three modes for **SELinux**
  - Enforcing
  - Permissive
  - Disabled
- Configured in `/etc/sysconfig/selinux` (**CentOS** and **openSUSE**) or `/etc/selinux/config` (**Ubuntu**)
- Use **getenforce** and **setenforce** to see or set current mode

- **Enforcing:** All **SELinux** code is operative and access is denied according to policy. All violations are audited and logged.
- **Permissive:** Enables **SELinux** code but only audits and warns about operations that would be denied in enforcing mode.
- **Disabled:** Completely disables **SELinux** kernel and application code leaving the system without any of its protections.

The **sestatus** utility can display the current mode and policy.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## getenforce and setenforce

- Examine or set the current mode:

- **getenforce**

```
$ getenforce
```

Enforcing

- **setenforce**

```
$ sudo setenforce Permissive
```

```
$ getenforce
```

Permissive

**setenforce** can be used to switch between enforcing and permissive mode on the fly while the system is in operation. However, changing to in or out of the disabled mode can not be done this way.

While **setenforce** allows you to switch between **Permissive** and **Enforcing** modes, it does not allow disabling **SELinux** completely. There are at least two different ways to disable **SELinux**:

- **Configuration file:** edit the **SELinux** configuration file (usually `/etc/selinux/config`) and set `SELINUX=disabled`. This is the default method and should be used to permanently disable **SELinux**.
- **Kernel parameter:** Adding `selinux=0` to the Kernel parameter list when rebooting.

However it is important to note that disabling **SELinux** on systems in which **SELinux** will be re-enabled is not recommended. It is preferable to use **Permissive** mode instead of disabling **SELinux**, so as to avoid relabeling the entire filesystem which can be time consuming.

# SELinux Policies

- **targeted:**

The **default** policy in which **SELinux** is more restrictive to targeted processes. User processes and **init** processes are not targeted, while network service processes are targeted. **SELinux** enforces memory restrictions for **all** processes, which reduces the vulnerability to buffer overflow attacks.

- **minimum:**

A modification of the targeted policy where only selected processes are protected.

- **MLS:**

The **Multi-Level Security** policy is much more restrictive; all processes are placed in fine-grained security domains with particular policies.

The same configuration file that sets the mode, usually `/etc/sysconfig/selinux`, also sets the **SELinux policy**.

Multiple policies are allowed, but only one can be active at a time.

Changing the policy may require a reboot of the system and a time-consuming re-labeling of filesystem contents.

Each policy has files which must be installed under `/etc/selinux/[SELINUXTYPE]`.

# Context Utilities

- Contexts and Context Utilities
  - User
  - Role
  - Type
  - Level
- Look Up or Change a Context:
  - Use -Z option to see context
 

```
$ ls -Z
$ ps auZ
```
  - Use **chcon** to change context
 

```
$ chcon -t etc_t somefile
$ chcon --reference somefile someotherfile
```

As mentioned earlier, contexts are labels applied to files, directories, ports, and processes. Those labels are used to describe access rules. There are four **SELinux** contexts, **User**, **Role**, **Type**, and **Level**.

We will focus on **type**, the most commonly utilized context. The label naming convention determines that type context labels should end with \_t as in kernel\_t.

```
$ ls -Z
1 -rw-rw-r--. dog dog unconfined_u:object_r:user_home_t:s0 somefile
```

```
$ chcon -t etc_t somefile
$ ls -Z
1 -rw-rw-r--. dog dog unconfined_u:object_r:etc_t:s0 somefile
```

```
$ ls -Z
1 -rw-rw-r--. dog dog unconfined_u:object_r:etc_t:s0 somefile
2 -rw-rw-r--. dog dog unconfined_u:object_r:user_home_t:s0 somefile1
```

```
$ chcon --reference somefile somefile1
$ ls -Z
1 -rw-rw-r--. dog dog unconfined_u:object_r:etc_t:s0 somefile
2 -rw-rw-r--. dog dog unconfined_u:object_r:etc_t:s0 somefile1
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

# SELinux and Standard Commands

- Many basic utilities (**ls**, **ps**, **cp**, etc., extended to work with **SELinux**
- Usually add Z option as in:

```
$ ls -Z ...
$ ps axZ
```

- With **SELinux** disabled, extended options do nothing

Many standard command line commands, such as **ls** and **ps**, were extended to support **SELinux**, and corresponding sections were added to their **man** pages explaining the details. Often the parameter Z is passed to standard command line tools as in:

```
$ ps axZ
1 LABEL PID TTY STAT TIME COMMAND
2 system_u:system_r:init_t:s0 1 ? Ss 0:04 /usr/lib/systemd/systemd --switched-root ...
3 system_u:system_r:kernel_t:s0 2 ? S 0:00 [kthreadd]
4 ...
5 unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 2305 ? D 0:00 sshd: jimih@pts/0
6 unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 2306 pts/0 Ss 0:00 -bash
7 ...
8 system_u:system_r:httpd_t:s0 7490 ? Ss 0:00 /usr/sbin/httpd -DFOREGROUND
9 system_u:system_r:httpd_t:s0 7491 ? S 0:00 /usr/sbin/httpd -DFOREGROUND
10 ...
```

```
$ ls -Z /home/ /tmp/
1 /home/:
2 drwx-----. jimih jimih unconfined_u:object_r:user_home_dir_t:s0 jimih
3 /tmp/:
4 -rwx-----. root root system_u:object_r:initrc_tmp_t:s0 ks-script-c4ENhg
5 drwx-----. root root system_u:object_r:tmp_t:s0 systemd-private-0ofSv0
6 -rw-----. root root system_u:object_r:initrc_tmp_t:s0 dnf.log
```

Other tools that were extended to support **SELinux** include **cp**, **mv**, and **mkdir**.

Note that if you have disabled **SELinux**, no useful information is displayed in the related fields from these utilities.

**For the exclusive use of LFS301 corp class taught 06 to 09 December**

## Context Inheritance

- New files inherit context from parent directory
- When moving files, source directory context preserved
- Can cause problems if context is not modified

Newly created files inherit the context from their parent directory, but when moving or copying files, it is the context of the source directory which may be preserved, which can cause problems.

Continuing the previous example, we see the context of `tmpfile` was not changed by moving the file from `/tmp` to `/home/jimih`:

```
$ cd /tmp/
$ touch tmpfile
$ ls -Z tmpfile
1 -rw-rw-r--. jimih jimih unconfined_u:object_r:user_tmp_t:s0 tmpfile
```

```
$ cd
$ touch homefile
$ ls -Z homefile
1 -rw-rw-r--. jimih jimih unconfined_u:object_r:user_home_t:s0 homefile
```

```
$ mv /tmp/tmpfile .
$ ls -Z
1 -rw-rw-r--. jimih jimih unconfined_u:object_r:user_home_t:s0 homefile
2 -rw-rw-r--. jimih jimih unconfined_u:object_r:user_tmp_t:s0 tmpfile
```

The classical example in which moving files creates an **SELinux** issue is moving files to the DocumentRoot directory of the **httpd** server. On **SELinux**-enabled systems, the web server can only access files with the correct context labels. Creating a file in `/tmp`, and then moving it to the DocumentRoot directory, will make the file inaccessible to the **httpd** server until the **SELinux** context of the file is adjusted.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# restorecon

- Reset file context in accord with parent directory:

```
$ restorecon -Rv /home/jimih
```

**restorecon** resets file contexts, based on parent directory settings.

In the following example, **restorecon** resets the default label recursively for all files at the home directory:

```
$ ls -Z
```

```
1 -rw-rw-r--. jimih jimih unconfined_u:object_r:user_home_t:s0 homefile
2 -rw-rw-r--. jimih jimih unconfined_u:object_r:user_tmp_t:s0 tmpfile
```

```
$ restorecon -Rv /home/jimih
```

```
1 restorecon reset /home/jimih/tmpfile context \
2 unconfined_u:object_r:user_tmp_t:s0->unconfined_u:object_r:user_home_t:s0
```

```
$ ls -Z
```

```
1 -rw-rw-r--. jimih jimih unconfined_u:object_r:user_home_t:s0 homefile
2 -rw-rw-r--. jimih jimih unconfined_u:object_r:user_home_t:s0 tmpfile
```

Note that the context for `tmpfile` has been reset to the default context for files created at the home directory. The type was changed from `user_tmp_t` to `user_home_t`.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# semanage

- Configure default context for a new directory
- Changes only default settings, not existing ones
- **restorecon** must be called afterwards to change existing contexts

Another issue is how to configure the default context for a newly created directory. **semanage fcontext** (provided by the **policycoreutils-python** package) can change and display the default context of files and directories. Note that **semanage fcontext** only changes the default settings; it does not apply them to existing objects. This requires calling **restorecon** afterwards. For example:

```
[root@rhel7 /]# mkdir /virtualHosts
[root@rhel7 /]# ls -Z
1 ...
2 drwxr-xr-x. root root unconfined_u:object_r:default_t:s0 virtualHosts
```

```
[root@rhel7 /]# semanage fcontext -a -t httpd_sys_content_t /virtualHosts
[root@rhel7 /]# ls -Z
1 ...
2 drwxr-xr-x. root root unconfined_u:object_r:default_t:s0 virtualHosts
```

```
[root@rhel7 /]# restorecon -RFv /virtualHosts
1 restorecon reset /virtualHosts context \
2 unconfined_u:object_r:default_t:s0->system_u:object_r:httpd_sys_content_t:s0
```

```
[root@rhel7 /]# ls -Z
1 drwxr-xr-x. root root system_u:object_r:httpd_sys_content_t:s0 virtualHosts
```

The context change from `default_t` to `httpd_sys_content_t` is thus only applied after the call to **restorecon**.  
**For the exclusive use of LFS301 corp class taught 06 to 09 December**

# Using SELinux Booleans

- Change the behavior of **SELinux** policy
- Can be enabled or disabled
  - **getsebool**: to see booleans
  - **setsebool**: to set booleans
  - **semanage boolean -l**: to see persistent boolean settings

```
$ setsebool allow_ftpd_anon_write on
$ getsebool allow_ftpd_anon_write
allow_ftpd_anon_write -> on
$ semanage boolean -l | grep allow_ftpd_anon_write
$ allow_ftpd_anon_write -> off
```

Note not persistent.

```
$ setsebool -P allow_ftpd_anon_write on
$ semanage boolean -l | grep allow_ftpd_anon_write
allow_ftpd_anon_write -> on
```

Now persistent.

| SELinux boolean                             | State       | Default | Description                                          |
|---------------------------------------------|-------------|---------|------------------------------------------------------|
| ftp_home_dir                                | (on , on)   |         | Allow ftp to home dir                                |
| smartmont3ware                              | (off , off) |         | Allow smartmont to 3ware                             |
| mpd_enable_homedirs                         | (off , off) |         | Allow mpd to enable homedirs                         |
| xdm_sysadm_login                            | (off , off) |         | Allow xdm to sysadm login                            |
| xen��                                       | (off , off) |         | Allow xen��                                          |
| mozilla_read_content                        | (off , off) |         | Allow mozilla to read content                        |
| ssh_chroot_rw_homedirs                      | (off , off) |         | Allow ssh to chroot rw homedirs                      |
| mount_anyfile                               | (on , on)   |         | Allow mount to anyfile                               |
| cron_userdomain_transition                  | (on , on)   |         | Allow cron to userdomain transition                  |
| icecast_use_any_tcp_ports                   | (off , off) |         | Allow icecast to use any tcp ports                   |
| openvpn_connect                             | (on , on)   |         | Allow openvpn to connect network connect             |
| anon_write                                  | (off , off) |         | Allow anon to write                                  |
| minidlna_read_generic_user_content          | (off , off) |         | Allow minidlna to read generic user content          |
| spamassassin_can_network                    | (off , off) |         | Allow spamassassin to can network                    |
| gluster_anon_write                          | (off , off) |         | Allow gluster to anon write                          |
| deny_ptrace                                 | (off , off) |         | Allow deny to ptrace                                 |
| selinuxuser_execmod                         | (on , on)   |         | Allow selinuxuser to execmod                         |
| httpd_use_network_relay                     | (off , off) |         | Allow httpd to use network relay                     |
| openvpn_enable_homedirs                     | (on , on)   |         | Allow openvpn to enable homedirs                     |
| glance_use_execmem                          | (off , off) |         | Allow glance to use execmem                          |
| telepathy_tcp_connect_generic_network_ports | (on , on)   |         | Allow telepathy to tcp connect generic network ports |
| httpd_can_connect_mythtv                    | (off , off) |         | Allow httpd to connect mythtv                        |

Figure 41.3: **semanage**

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Monitoring SELinux Access

- Use **setroubleshoot-server** to track SELinux errors
  - After install, restart **auditd** service
- Stores raw messages in `/var/log/audit/audit.log`
- Moves messages to `/var/log/messages`
- Use **sealert** to see detailed message

**SELinux** comes with a set of tools that collect issues at run time, log these issues and propose solutions to prevent same issues from happening again. These utilities are provided by the **setroubleshoot-server** package. Here is an example of their use:

```

1 [root@rhel7 ~]# echo 'File created at /root' > rootfile
2 [root@rhel7 ~]# mv rootfile /var/www/html/
3 [root@rhel7 ~]# wget -O - localhost/rootfile
4 --2014-11-21 13:42:04-- http://localhost/rootfile
5 Resolving localhost (localhost)... ::1, 127.0.0.1
6 Connecting to localhost (localhost)|::1|:80... connected.
7 HTTP request sent, awaiting response... 403 Forbidden
8 2014-11-21 13:42:04 ERROR 403: Forbidden.
9
10 [root@rhel7 ~]# tail /var/log/messages
11 Nov 21 13:42:04 rhel7 setroubleshoot: Plugin Exception restorecon
12 Nov 21 13:42:04 rhel7 setroubleshoot: SELinux is preventing /usr/sbin/httpd from setattr access on the file .
13 For complete SELinux messages. run sealert -l d51d34f9-91d5-4219-ad1e-5531e61a2dc3
14 Nov 21 13:42:04 rhel7 python: SELinux is preventing /usr/sbin/httpd from setattr access on the file .
15
16 Do allow this access for now by executing:
17 # grep httpd /var/log/audit/audit.log | audit2allow -M mypol
18 # semodule -i mypol.pp
19
20 Additional Information:
21 Source Context system_u:system_r:httpd_t:s0
22 Target Context unconfined_u:object_r:admin_home_t:s0
23 Target Objects [file]
24 Source httpd
25 Source Path /usr/sbin/httpd
26

```

For the exclusive use of LFS301 corp class taught 06 to 09 December

### 41.3 AppArmor

## AppArmor

- Provides Mandatory Access Control (**MAC**).
- Allows administrators to associate a security profile to a program which restricts its capabilities.
- Is considered easier (by some but not all) to use than **SELinux**.
- Is considered filesystem-neutral (no security labels required).

**AppArmor** is an **LSM** alternative to **SELinux**. Support for it has been incorporated in the **Linux** kernel since 2006. It has been used by **SUSE**, **Ubuntu** and other distributions.

**AppArmor** supplements the traditional **UNIX Discretionary Access Control (DAC)** model by providing **Mandatory Access Control (MAC)**.

In addition to manually specifying profiles, **AppArmor** includes a learning mode, in which violations of the profile are logged, but not prevented. This log can then be turned into a profile, based on the program's typical behavior.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Checking Status

- **AppArmor** usually enabled and loaded by default
- **Linux** kernel must have it compiled and turned on
- Start and stop etc:
 

```
$ sudo systemctl [start|stop|restart|status] apparmor
```
- Cause to be loaded or not loaded at boot:
 

```
$ sudo systemctl [enable|disable] apparmor
```
- More detail on current status:
 

```
$ sudo apparmor_status
```
- **Profiles** and **processes** in either **enforce** or **complain** mode
  - Directly analogous to **enforcing** and **permissive** modes in **SELinux**

Distributions that come with **AppArmor**, tend to enable it and load it by default; note that the **Linux** kernel has to have it turned on as well, and in most cases only one **LSM** can run at a time.

Assuming you have the **AppArmor** kernel module available, on a **systemd** equipped system you can do:

```
$ sudo systemctl [start|stop|restart|status] apparmor
```

to change or inquire about the current state of operation, or do:

```
$ sudo systemctl [enable|disable] apparmor
```

to cause to be loaded or not loaded at boot.

In order to see the current status:

```
$ sudo apparmor_status
```

```

1 apparmor module is loaded.
2 25 profiles are loaded.
3 25 profiles are in enforce mode.
4 /sbin/dhclient
5
```

**Profiles** and **processes** are in either **enforce** or **complain** mode, directly analogous to **SELinux**'s **enforcing** and **permissive**.

Note in the process listing the **PID** is given:

```
$ ps aux | grep libvirt
```

```

1 root 787 0.0 0.9 527200 35936 ? Ssl 10:54 0:00 /usr/sbin/libvirt
2 student 3346 0.0 0.0 13696 2204 pts/16 S+ 11:42 0:00 grep --color=auto libvirt
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Modes and Profiles

- Modes:
  - **Enforce Mode** Restricted actions are prevented and reported. Set with **aa-enforce**
  - **Complain** Policies not enforced, but policy violations are reported. Set with **aa-complain**
- **Profiles** configured in `/etc/apparmor.d` give details for utilities and software packages

**Profiles** restrict how executable programs, which have pathnames on your system, such as `/usr/bin/evince`, can be used. Processes can be run in either of two modes:

- **Enforce Mode**

Applications are prevented from acting in ways which are restricted. Attempted violations are reported to the system logging files. This is the default mode. A profile can be set to this mode with **aa-enforce**.

- **Complain**

Polices are not enforced, but attempted policy violations are reported. This is also called **learning** mode. A profile can be set to this mode with **aa-complain**

Linux distributions come with pre-packaged profiles, typically installed either when a given package is installed, or with an AppArmor package, such as **apparmor-profiles**. These profiles are stored in `/etc/apparmor.d`.

When installing new software, new profiles can be created specific to any executables in the package.

Exactly what AppArmor profiles are installed on your system depends on your selection of software packages. On one particular Ubuntu system:

```
student@ubuntu:/etc/apparmor.d$ ls
```

|                |                        |                       |
|----------------|------------------------|-----------------------|
| 1 abstractions | usr.lib.dovecot.anvil  | usr.lib.telepathy     |
| 2 apache2.d    | usr.lib.dovecot.auth   | usr.sbin.avahi-daemon |
| 3 bin.ping     | usr.lib.dovecot.config | usr.sbin.cups-brows   |
| 4 ....         |                        |                       |

Full documentation on what can go in these files can be obtained by doing `man apparmor.d`.

**For the exclusive use of LFS301 corp class taught 06 to 09 December**

# AppArmor Utilities

Table 41.1: AppArmor Utilities

| Program                | Use                                                                                                                                                                       |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>apparmor_status</b> | Show status of all profiles and processes with profiles                                                                                                                   |
| <b>apparmor_notify</b> | Show a summary for AppArmor log messages                                                                                                                                  |
| <b>complain</b>        | Set a specified profile to complain mode.                                                                                                                                 |
| <b>enforce</b>         | Set a specified profile to enforce mode.                                                                                                                                  |
| <b>disable</b>         | Unload a specified profile from the current kernel and prevent from being loaded on system startup.                                                                       |
| <b>logprof</b>         | Scan log files and if AppArmor events have been recorded that are not covered by existing profiles, suggest how to take into account, and if approved, modify and reload. |
| <b>easyprof</b>        | Help set up a basic AppArmor profile for a program.                                                                                                                       |

AppArmor has quite a few administrative utilities for monitoring and control. For example, on an openSUSE system:

```
$ rpm -qil apparmor-utils | grep bin
```

```

1 /usr/bin/aa-easyprof
2 /usr/sbin/aa-audit
3 /usr/sbin/aa-autodep
4 /usr/sbin/aa-cleanprof
5 /usr/sbin/aa-complain
6 /usr/sbin/aa-decode
7 /usr/sbin/aa-disable
8 /usr/sbin/aa-enforce
9 /usr/sbin/aa-exec
10
11 /usr/sbin/complain
12 /usr/sbin/decode
13 /usr/sbin/disable
14 /usr/sbin/enforce
15

```

Note that many of these utilities can be invoked with either their short or long names; e.g.:

```
linux-llgn:/etc/apparmor.d # ls -l /usr/sbin/*complain
1 -rwxr-xr-x 1 root root 1442 Oct 25 07:37 /usr/sbin/aa-complain*
2 lwxrwxrwx 1 root root 11 Nov 11 13:02 /usr/sbin/complain -> aa-complain*
3 linux-llgn:/etc/apparmor.d #
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 41.4 Labs

### Exercise 41.1: SELinux: Contexts



#### Please Note

This exercise can only be performed on a system (such as **RHEL**) where **SELinux** is installed. While it is possible to install on **Debian**-based distributions, such as **Ubuntu**, it is not the easiest task and it is not often done.

1. Verify **SELinux** is enabled and in **enforcing** mode, by executing **getenforce** and **sestatus**. If not, edit **/etc/selinux/config**, reboot, and check again.
2. Install the **httpd** package (if not already present) which provides the **Apache** web server, and then verify that it is working:

```
$ sudo dnf install httpd
$ sudo systemctl start httpd
$ elinks http://localhost
```

(You can also use **lynx** or **elinks** etc. as the browser, or use your graphical browser such as **firefox** or **chrome**, in this and succeeding steps.)

3. As superuser, create a small file in **/var/www/html**:

```
$ sudo sh -c "echo file1 > /var/www/html/file1.html"
```

4. Verify you can see it:

```
$ elinks -dump http://localhost/file1.html
```

```
1 file1
```

Now create another small file in **root**'s home directory and **move** it to **/var/www/html**. (Do not copy it, move it!) Then try and view it:

```
$ sudo cd /root
$ sudo sh -c "echo file2 > file2.html"
$ sudo mv file2.html /var/www/html
$ elinks -dump http://localhost/file2.html
```

```
1
2
3
Forbidden
You don't have permission to access /file2.html on this server.
```

5. Examine the security contexts:

```
$ cd /var/www/html
$ ls -Z file*html
1 -rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 file1.html
2 -rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 file2.html
```

6. Change the offending context and view again:

```
$ sudo chcon -t httpd_sys_content_t file2.html
$ elinks http://localhost/file2.html
```

```
1 file2
```

### Exercise 41.2: Exploring apparmor security

For the exclusive use of LFS301 corp class taught 06 to 09 December

**Please Note**

This exercise can only be performed on a system (such as **Ubuntu**) where **AppArmor** is installed.

The below was tested on **Ubuntu**, but should work on other **AppArmor**-enabled systems, such as **openSUSE**, where the **apt-get** commands should be replaced by **zypper**.

On **Ubuntu**, the **/bin/ping** utility runs with SUID enabled. For this exercise, we will copy **ping** to **ping-x** and adjust the capabilities so the program functions.

Then we will build an **AppArmor** profile, install and verify that nothing has changed. Modifying the **AppArmor** profile and adding capabilities will allow the program more functionality.

1. Make sure all necessary packages are installed:

```
student@ubuntu:~$ sudo apt-get install apparm*
```

2. Create a copy of **ping** (called **ping-x**) and verify it has no initial special permissions or capabilities. Furthermore, it cannot work when executed by **student**, a normal user:

```
student@ubuntu:~$ sudo cp /bin/ping /bin/ping-x
student@ubuntu:~$ sudo ls -l /bin/ping-x
-rwxr-xr-x 1 root root 64424 Oct 17 10:12 /bin/ping-x

student@ubuntu:~$ sudo getcap /bin/ping-x
student@ubuntu:~$

student@ubuntu:~$ ping-x -c3 -4 127.0.0.1
ping: socket: Operation not permitted
```

3. Set the **capabilities** and re-try **ping-x**:

```
student@ubuntu:~$ sudo setcap cap_net_raw+ep /bin/ping-x

student@ubuntu:~$ ping-x -c3 -4 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.092 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.093 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.086 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2034ms
rtt min/avg/max/mdev = 0.086/0.090/0.093/0.008 ms
```

The modified **ping-x** program now functions normally.

4. Verify there is no pre-existing **AppArmor** profile for **ping-x**, but there is a profile for **ping**. Determine the status of the current **ping** program:

```
student@ubuntu:~$ sudo aa-status
```

The output from **aa-status** is long, so we can **grep** for the interesting lines:

```
student@ubuntu:~$ sudo aa-status | grep -e "^[[:alnum:]]" -e ping

apparmor module is loaded.
87 profiles are loaded.
51 profiles are in enforce mode.
 ping
36 profiles are in complain mode.
17 processes have profiles defined.
6 processes are in enforce mode.
11 processes are in complain mode.
0 processes are unconfined but have a profile defined.
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

We can see **ping** has a profile that is loaded and enabled for enforcement.

5. Next we will construct a new profile for **ping-x**. This step requires two terminal windows.

The first window (**window1**) will be running the **aa-genprof** command. This will generate a **AppArmor** profile by scanning **/var/log/syslog** for **AppArmor** errors.

The second window (**window2**) will be used to run **ping-x**. (See the **man** page for **aa-genprof** for additional information.)

In **window1**:

```
student@ubuntu:~$ sudo aa-genprof /bin/ping-x
Writing updated profile for /bin/ping-x.
Setting /bin/ping-x to complain mode.
```

Before you begin, you may wish to check if a profile already exists for the application you wish to confine. See the following wiki page for more information:  
<http://wiki.apparmor.net/index.php/Profiles>

Please start the application to be profiled in another window and exercise its functionality now.

Once completed, select the "Scan" option below in order to scan the system logs for AppArmor events.

For each AppArmor event, you will be given the opportunity to choose whether the access should be allowed or denied.

Profiling: **/bin/ping-x**

**[(S)can system log for AppArmor events] / (F)inish**

In **window2**:

```
student@ubuntu:~$ ping-x -c3 -4 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.099 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.120 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.114 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2041ms
rtt min/avg/max/mdev = 0.099/0.111/0.120/0.008 ms
```

In **window1**:

The command **ping-x** has completed, we must now instruct **aa-genprof** to scan for the required information to be added to the profile. It may require several **scans** to collect all of the information for the profile.

Enter S to scan:

```
Reading log entries from /var/log/syslog.
Updating AppArmor profiles in /etc/apparmor.d.
Complain-mode changes:
```

```
Profile: /bin/ping-x
Capability: net_raw
Severity: 8

[1 - capability net_raw,]
(A)llow / [(D)eny] / (I)gnore / Audi(t) / Abo(r)t / (F)inish
```

Enter A to allow the capability:

**For the exclusive use of LFS301 corp class taught 06 to 09 December**

```
Adding capability net_raw, to profile.

Profile: /bin/ping-x
Network Family: inet
Socket Type: raw

[1 - network inet raw,]
(A)llow / [(D)eny] / (I)gnore / Audi(t) / Abo(r)t / (F)inish
```

Enter A to allow the network family:

```
Adding network inet raw, to profile.
```

```
Profile: /bin/ping-x
Network Family: inet
Socket Type: dgram

[1 - #include <abstractions/nameservice>]
2 - network inet dgram,
(A)llow / [(D)eny] / (I)gnore / Audi(t) / Abo(r)t / (F)inish
```

Enter A to add the socket type datagram to the profile:

```
Adding #include <abstractions/nameservice> to profile.
```

```
= Changed Local Profiles =
```

```
The following local profiles were changed. Would you like to save them?
```

```
[1 - /bin/ping-x]
(S)ave Changes / Save Selec(t)ed Profile / [(V)iew Changes] / View Changes b/w (C)lean profiles / Abo(r)t
```

Enter S to save the new profile:

```
Writing updated profile for /bin/ping-x.
```

```
Profiling: /bin/ping-x
```

```
[(S)can system log for AppArmor events] / (F)inish
```

Enter F to finish:

```
Setting /bin/ping-x to enforce mode.
```

```
Reloaded AppArmor profiles in enforce mode.
```

```
Please consider contributing your new profile!
```

See the following wiki page for more information:

<http://wiki.apparmor.net/index.php/Profiles>

```
Finished generating profile for /bin/ping-x.
```

- View the created profile, which has been stored in `/etc/apparmor.d/bin.ping-x`.

```
student@ubuntu:~$ sudo cat /etc/apparmor.d/bin.ping-x
Last Modified: Tue Oct 17 11:30:47 2017
#include <tunables/global>

/bin/ping-x {
 #include <abstractions/base>
 #include <abstractions/nameservice>

 capability net_raw,

 network inet raw,
```

**For the exclusive use of LFS301 corp class taught 06 to 09 December**

```

/bin/ping-x mr,
/lib/x86_64-linux-gnu/ld-* .so mr,
}

```

- The **aa-genproc** utility installs and activates the new policy so it should be ready to use, and the policies can be reloaded on demand with the `systemctl reload apparmor` command. To avoid any potential issues, and verify the changes will survive, reboot the system.

Once the system has restarted, as the user student, verify **ping-x** still functions with the new profile enabled. Ping the localhost by ip address:

```

student@ubuntu:~$ ping-x -c3 -4 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.057 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.043 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.095 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2027ms
rtt min/avg/max/mdev = 0.043/0.065/0.095/0.021 ms

```

- This should work as expected. The profile is very specific, and **AppArmor** will not allow functionality outside of the specified parameters. To verify **AppArmor** is protecting this application, try to ping the **IPv6** localhost address.

This should fail:

```

student@ubuntu:~$ ping-x -c3 -6 ::1
ping: socket: Permission denied

```

(Note, the **-6** option means use only **IPv6** and **::1** is the local host in **IPv6**.)

The output indicates there is a socket issue. If the system log is examined it will be discovered that our **ping-x** program has no access to **IPv6** within **AppArmor**:

```

766:104): apparmor="DENIED" operation="create" profile="/bin/ping-x"
pid=2709 comm="ping-x" family="inet6" sock_type="raw" protocol=58
requested_mask="create" denied_mask="create"

```

- To correct this deficiency, re-run **aa-genprof** as we did earlier, and in **window2**, ping the **IPv6** loopback and append the additional options

## Chapter 42

# Local System Security



|      |                                      |     |
|------|--------------------------------------|-----|
| 42.1 | Local System Security . . . . .      | 540 |
| 42.2 | Creating a Security Policy . . . . . | 541 |
| 42.3 | Updates and Security . . . . .       | 545 |
| 42.4 | Physical Security . . . . .          | 546 |
| 42.5 | BIOS . . . . .                       | 548 |
| 42.6 | Bootloader . . . . .                 | 549 |
| 42.7 | Filesystem Security . . . . .        | 550 |
| 42.8 | setuid/setgid bits . . . . .         | 551 |
| 42.9 | Labs . . . . .                       | 552 |

## 42.1 Local System Security

# Local System Security

- Need to secure against both **internal** and **external** threats
- Threats may be expected, or unexpected
- Local security policy must be carefully designed and carried through
- Systems should be properly maintained and updated
- Systems should be physically secured
- Only appropriate users should have potentially dangerous privileges, only those absolutely needed
- All users should be aware of security policies and good security hygiene, and rules must be enforced

Computers are inherently insecure and need protection from the people who would intrude in on or attack them. This can be done to simply harm the system, deny services, or steal information.

No computer can ever be absolutely secure. All we can do is slow down and/or discourage intruders so that they either go away and hunt for easier targets, or so we can catch them in the act and take appropriate action.

Security can be defined in terms of the system's ability to regularly do what it is supposed to do, integrity and correctness of the system, and ensuring that the system is only available to those authorized to use it.

The biggest problem with security is to find that appropriate mix of security and productivity; if security restrictions are tight, opaque, and difficult, especially with ineffective measures, users will circumvent procedures.

The four areas we need to protect include physical, local, remote, and personnel. In this section we will not concentrate on network security, we will concentrate on the local factors.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 42.2 Creating a Security Policy

# Creating a Security Policy

The security policy should:

- Be simple and easy to understand
- Get constantly updated
- Be in the form of a written document in addition to online documentation if needed
- Describe both policies and procedures.
- Specify enforcement actions
- Specify actions to take in response to a security breach

Policies should be generic and not hard to grasp as that makes them easier to follow. They must safeguard the data that needs protection, deny access to required services and protect user privacy.

These policies should be updated on a regular basis; policies need to change as requirements do. Having an out of date policy can be worse than having none.

# What to Include in the Policy

- Confidentiality
- Data Integrity
- Availability
- Consistency
- Control
- Audit

A security policy should include methods of protecting information from being read or copied by unauthorized personnel. It should also include protection of information from being altered or deleted without the permission of the owner. All services should be protected so they are available and not degraded in any manner without authorization.

You should make sure that the data is correct and the system behaves as it is expected to do. There should be processes in effect to determine who is given access to your system. The human factor is the weakest link in the security chain; it requires the most attention through constant auditing.

---

For the exclusive use of LFS301 corp class taught 06 to 09 December

## What Risks to Assess

- What do I want to protect?
- What am I protecting against?
- How much time, personnel, and money is needed to provide adequate protection?

What is to be protected (identify assets), what is being protected against (identify threats), and what will it take to protect the system are the basis of Risk Analysis. Risk analysis is the first step in computer security.

You must know what you are protecting and what you are protecting against in order to determine how to protect your systems. This allows you to then plan policies and procedures to protect your systems.

This is the first step taken in constructing a computer security policy. It is a pre-requisite for planning and then enforcing policies and procedures to protect your systems.

For the exclusive use of LFS301 corp class taught 06 to 09 December

# Choosing a Security Philosophy

- There are basically two philosophies in use in most computing environments
  - Anything not expressly permitted is denied
  - Anything not expressly forbidden is permitted
- Remember:
  1. The human factor is the weakest link
  2. No computing environment is invulnerable
  3. Paranoia is a good thing

The first choice is tighter: a user is allowed to do only what is clearly and explicitly specified as permissible without privilege. This is the most commonly used philosophy.

The second choice builds a more liberal environment where users are allowed to do anything except what is expressly forbidden. It implies a high degree of assumed trust and is less often deployed for obvious reasons.

Some general guidelines to remember when developing security philosophies are:

**1. The human factor is the weakest link.**

You must educate your users and keep them happy. The largest percentage of break ins are internal and are not often malicious.

**2. No computing environment is invulnerable.**

The only secure system is one that is not connected to anything, locked in a secure room and turned off.

**3. Paranoia is a good thing.**

Be suspicious, be vigilant and persevere when securing a computer. It is an ongoing process which must be constantly paid attention to. Check process, users, and look for anything out of the ordinary.

Users should never put the current directory in their path, i.e, do not do something in `~/.bashrc` like: `PATH=.:$PATH` This is a significant security risk; a malicious person could substitute for a program with one of the same name, that could do harmful things. Just think of a script named `ls` that contains just the line: `/bin/rm -rf $HOME` If you were to go to the directory that contains this file and type `ls`, you would wipe out your home directory!

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 42.3 Updates and Security

### Updating and Patching the System

- Security flaws are constantly found
- Most are very minor or have no known exploits
- A system which is not fully updated is vulnerable
- One of the most important jobs of the **Linux** distribution is to disseminate security fixes rapidly and carefully
- Examples of an update hurting a **Linux** system are very rare, especially as compared to other operating systems.

It is critical to pay attention to your **Linux** distributor's updates and upgrades and apply them as soon as possible.

Most attacks exploit known security holes and are deployed in the time period between revelation of a problem and patches being applied. **Zero Day** exploits are actually much rarer, where an attacker uses a security hole that either has not been discovered yet or for which a fix has not been released.

System administrators are sometimes reluctant to apply such fixes immediately upon release, based on negative experiences with proprietary operating system vendors who can cause more problems with fixes than they solve. However, in **Linux** such security **regressions** are extremely rare, and the danger of delaying applying a security patch is probably never justifiable.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 42.4 Physical Security

### Hardware Accessibility Vulnerabilities

Hardware which is physically accessible can be compromised by:

- Key logging: Recording the real time activity of a computer user including the keys they press. The captured data can either be stored locally or transmitted to remote machines.
- Network sniffing: Capturing and viewing the network packet level data on your network.
- Booting with a live or rescue disk.
- Remounting and modifying disk content.

Physical access to a system makes it possible for attackers to easily leverage several attack vectors, in a way that makes all operating system level recommendations irrelevant.

Thus, security policy should start with requirements on how to properly secure physical access to servers and workstations.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Hardware Access Guidelines

Protective steps include:

- Locking down workstations and servers.
- Protecting your network links against access by people you do not trust.
- Protecting your keyboards where passwords are entered to ensure the keyboards cannot be tampered with.
- Configuring password protection of the **BIOS** in such a way that the system cannot be booted with a live or rescue **CD/DVD** or **USB** key.

For single user computers and those in a home environment, some of the above features (like preventing booting from removable media) can be excessive, and you can avoid implementing them. However, if sensitive information is on your system that requires careful protection, either it should not be there, or it should be better protected by following the above guidelines.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 42.5 BIOS

# BIOS

- Lowest level of security
- Use **BIOS** password
- Keep updated

The **BIOS** is the lowest level of software that configures or manipulates your system. The bootloader accesses the **BIOS** to determine how to boot up your machine.

Setting a **BIOS** password protects against unauthorized persons changing the boot options to gain access to your system. However, it only matters if someone can gain physical access to the machine as it requires a local presence.

Also, it is generally recommended that the **BIOS** be kept patched to the latest version of the firmware. However, most **BIOS** updates have nothing to do with security and system administrators have also been instructed to apply new **BIOS** only with care, as incompetent **BIOS** code has always been a plague, and unnecessary updates can render a system useless.

## 42.6 Bootloader

# Bootloader

- Use bootloader password
  - Can be global or local (to a stanza)
- Work in conjunction with **BIOS** password
  - Will not stop attack with removable bootable media

You can secure the boot process with a secure password to prevent someone from bypassing the user authentication step. This can work in conjunction with password protection for the **BIOS**.

Note that while using a bootloader password alone will stop a user from editing the bootloader configuration during the boot process, it will **not** prevent a user from booting from an alternative boot media such as optical disks or pen drives. Thus, it should be used with a **BIOS** password for full protection.

For the older **GRUB 1**, it was relatively easy to set a password for **grub**, but for the dominant **GRUB 2** version, things are more complicated. However, you have more flexibility and can do things like setting individual user-specific passwords (which can be the normal login ones.)

Once again you should not edit `grub.cfg` directly. Rather you edit system configuration files in `/etc/grub.d` and then run **update-grub** or **grub2-mkconfig** and save the new configuration file.

One explanation of this can be found at <https://help.ubuntu.com/community/Grub2/Passwords>.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 42.7 Filesystem Security

### Using Secure Mounting Options

- `nodev`  
Do not interpret character or block special devices on the filesystem
- `nosuid`  
The **set-user-identifier** or **set-group-identifier** bits are not to take effect
- `noexec`  
Restrict direct execution of any binaries on the mounted filesystem
- `ro`  
Mount the filesystem in **read-only** mode

When a filesystem is mounted, either at the command line with a **mount** command, or automatically by inclusion in `/etc/fstab`, various options can be specified to enhance security

For example, the `ro` option is used to mount the filesystem in read-only mode.

```
$ mount -o ro,noexec,nodev /dev/sda2 /edsel
```

In `/etc/fstab`

```
/dev/sda2 /edsel ext4 ro,noexec,nodev 0 0
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 42.8 setuid/setgid bits

# setuid/setgid bits

- For executable files
  - Process run by user have privileges of owner or group

```
$ chmod u+s somefile
$ chmod g+s somefile
```
- For directories
  - Used to create a shared directory
  - Files created in **sgid** directory are group owned by the group owner of directory

```
$ chmod g+s somedir
```
- Changing the **setuid** bit on a script has no effect.
  - You would have to change it on **bash** itself, a very bad idea!

Normally, programs are run with the privileges of the user who is executing the program. This means no matter who actually owns the binary executable that is running, the process still has constricted privileges.

Occasionally it may make sense to have normal users have expanded capabilities they would not normally have, such as the ability to start or stop a network interface, or edit a file owned by the superuser.

By setting the **setuid** (set user ID) flag on an executable file, one modifies this normal behavior by giving the program the access rights of the **owner** rather than the **user** of the program.

Furthermore, one can also set the **setgid** bit so the process runs with the privileges of the group that owns the file rather than those of the one running it.

We should emphasize that this is generally a **bad idea** and is to be avoided in most circumstances. It is often better to write a **daemon** program with lesser privileges for this kind of use. Some **Linux** distributions have actually disabled this ability entirely.

By default, when a file is created in a directory it is owned by the user and group of the user that created it. Using the **setgid** setting on the directory changes this so that files created in the directory are group owned by the group owner of the directory. This allows you to create a shared directory in which a group of users can share files.

## 42.9 Labs

### Exercise 42.1: Security and Mount Options

We are going to mount a partition or loop device with the `noexec` option to prevent execution of programs that reside on the filesystem therein. You can certainly do this with a pre-existing and mounted partition, but you may not be able to easily change the behavior while the partition is mounted. Therefore, to demonstrate we'll use a loop device, which is a harmless procedure.

1. Set up an empty file, put a filesystem on it and mount it.
2. Copy an executable file to it from somewhere else on your system and test that it works in the new location.
3. Unmount it and remount with the `noexec` option.
4. Test if the executable still works. It should give you an error because of the `noexec` mount option.
5. Clean up.

### Solution 42.1

1. 

```
$ dd if=/dev/zero of=image bs=1M count=100
$ sudo mkfs.ext3 image
$ mkdir mountpoint
$ sudo mount -o loop image mountpoint
```
2. 

```
$ sudo cp /bin/ls mountpoint
$ mountpoint/ls
```
3. 

```
$ sudo umount mountpoint
$ sudo mount -o noexec,loop image mountpoint
```

or

```
$ sudo mount -o noexec,remount image mountpoint
```
4. 

```
$ mountpoint/ls
```
5. 

```
$ sudo umount mountpoint
$ rm image
$ rmdir mountpoint
```

Note that this is not persistent. To make it persistent you would need to add the option to `/etc/fstab` with a line like:



in `/etc/fstab`

```
/home/student/image /home/student/mountpoint ext3 loop,rw,noexec 0 0
```

### Exercise 42.2: More on setuid and Scripts

Suppose we have the following C program (`./writeit.c`) which attempts to overwrite a file in the current directory named `afile`. This file can be extracted from your downloaded SOLUTIONS file as `writeit.c`.

For the exclusive use of LFS301 corp class taught 06 to 09 December



## writeit.c

```

1 /*
2 */
3 #include <stdio.h>
4 #include <unistd.h>
5 #include <fcntl.h>
6 #include <stdlib.h>
7 #include <string.h>
8 #include <sys/stat.h>
9
10
11 int main(int argc, char *argv[])
12 {
13 int fd, rc;
14 char *buffer = "TESTING A WRITE";
15 fd = open("./afile", O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
16 rc = write(fd, buffer, strlen(buffer));
17 printf("wrote %d bytes\n", rc);
18 close(fd);
19 exit(EXIT_SUCCESS);
20 }
```

If the program is called `writeit.c`, it can be compiled simply by doing:

```
$ make writeit
```

or equivalently

```
$ gcc -o writeit writeit.c
```

If (as a normal user) you try to run this program on a file owned by root you'll get

```
$ sudo touch afile
$./writeit
```

```
1 wrote -1 bytes
```

but if you run it as root:

```
$ sudo ./writeit
```

```
1 wrote 15 bytes
```

Thus, the root user was able to overwrite the file it owned, but a normal user could not.

Note that changing the owner of `writeit` to root does not help:

```
$ sudo chown root.root writeit
$./writeit
```

```
1 wrote -1 bytes
```

because it still will not let you clobber `afile`.

By setting the **setuid** bit you can make any normal user capable of doing it:

```
$ sudo chmod +s writeit
$./writeit
```

```
1 wrote 15 bytes
```

For the exclusive use of LFS301 corp class taught 06 to 09 December



### Please Note

You may be asking, why didn't we just write a script to do such an operation, rather than to write and compile an executable program?

Under **Linux**, if you change the **setuid** bit on such an executable script, it won't do anything unless you actually change the **setuid** bit on the shell (such as **bash**) which would be a big mistake; anything running from then on would have escalated privilege!

---

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 43

# Basic Troubleshooting



|      |                                              |     |
|------|----------------------------------------------|-----|
| 43.1 | Troubleshooting Levels . . . . .             | 556 |
| 43.2 | Troubleshooting Techniques . . . . .         | 557 |
| 43.3 | Networking . . . . .                         | 558 |
| 43.4 | File Integrity . . . . .                     | 559 |
| 43.5 | Boot Process Failures . . . . .              | 560 |
| 43.6 | Filesystem Corruption and Recovery . . . . . | 561 |
| 43.7 | Virtual Consoles . . . . .                   | 562 |
| 43.8 | Labs . . . . .                               | 563 |

## 43.1 Troubleshooting Levels

# Troubleshooting Levels

Three Levels of Troubleshooting:

- **Beginner:** Can be taught very quickly.
- **Experienced:** Comes after a few years of practice.
- **Wizard:** Some people think you have to be born this way, but that is nonsense; all skills can be learned.

Even the best administered **Linux** systems will develop problems.

**Troubleshooting** can isolate whether the problems arise from software or hardware, as well as whether they are local to the system, or come from within the local network or the Internet.

Troubleshooting properly requires judgment and experience, and while it will always be somewhat of an art form, following good methodical procedures can really help isolate the sources of problems in a reproducible fashion.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 43.2 Troubleshooting Techniques

### Basic Troubleshooting Techniques

- Characterize the problem.
- Reproduce the problem.
- Always try the easy things first.
- Eliminate possible causes one at a time.
- Change only one thing at a time; if that does not fix the problem, change it back.
- Check the system logs (`/var/log/messages`, `/var/log/secure`, etc.) for further information.

Sometimes the ruling philosophy and methodology requires following a very established procedure; making leaps based on intuition is discouraged. The motivation for using a checklist and uniform procedure is to avoid reliance on a wizard, to ensure any system administrator will be able to eventually solve a problem if they adhere to well known procedures. Otherwise, if the wizard leaves the organization, there is no one skilled enough to solve tough problems.

If, on the other hand, you elect to respect your intuition and check hunches, you should make sure you can get sufficient data quickly enough to decide whether or not to continue or abandon an intuitive path, based on whether it looks like it will be productive.

While ignoring intuition can sometimes make solving a problem take longer, the troubleshooter's previous track record is the critical benchmark for evaluating whether to invest resources this way. In other words, useful intuition is not magic, it is distilled experience.

For the exclusive use of LFS301 corp class taught 06 to 09 December

### 43.3 Networking

## Things to Check: Networking

- IP configuration
- Network Driver
- Connectivity
- Default gateway and routing configuration
- Hostname resolution

Network problems can be caused either by software or hardware, and can be as simple as is the device driver loaded, or is the network cable connected. If the network is up and running but performance is terrible, it really falls under the banner of performance tuning, not troubleshooting. The problems may be external to the machine, or require adjustment of the various networking parameters including buffer sizes etc.

The following items need to be checked when there are issues with networking:

- **IP configuration:**

Use **ifconfig** or **ip** to see if the interface is up, and if so if it is configured

- **Network Driver:**

If the interface cannot be brought up, maybe the correct device driver for the network card(s) is not loaded. Check with **lsmod** if the network driver is loaded as a kernel module, or by examining relevant pseudo-files in **/proc** and **/sys**, such as **/proc/interrupts** or **/sys/class/net**.

- **Connectivity:**

Use **ping** to see if the network is visible, checking for response time and packet loss. **traceroute** can follow packets through the network, while **mtr** can do this in a continuous fashion. Use of these utilities can tell you if the problem is local or on the Internet.

- **Default gateway and routing configuration:**

Run **route -n** and see if the routing table make sense.

- **Hostname resolution:**

Run **dig** or **host** on a URL and see if **DNS** is working properly.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 43.4 File Integrity

### Things to Check: File Integrity

- Check for corrupt configuration files or binaries
- Can use numerous methods
  - **rpm**  
    \$ rpm -V some\_package  
    to check a single package  
    \$ rpm -Va  
    to check all packages
  - **aide**: intrusion detection  
    \$ aide --check
  - **debsums**: **Debian**  
    \$ debsums options some\_package

There are a number of ways to check for corrupt configuration files and binaries. One way is to use the command `rpm -V` to check a particular package. You can also use `rpm -Va` to check the integrity of all packages.

Using **aide** is another way to check for changes in files. The command `aide --check` will run a scan on your files and compare them to the last scan.

In **Debian**, the only way to do integrity checking is with **debsums**. Running `debsums somepackage` will check the checksums on the files in that package. However, not all packages maintain checksums so this might be less than useful.

## 43.5 Boot Process Failures

# Boot Process Failures

- Follow the boot process
  - BIOS
  - GRUB
  - Kernel
  - init

If the system fails to boot properly or fully, being familiar with what happens at each stage is important. Assuming you get through the **BIOS** stage, you may reach a state of:

### No bootloader screen:

Check for **GRUB** misconfiguration or a corrupt boot sector. Bootloader re-install needed?

### Kernel fails to load:

If the kernel **panics** during the boot process, it is most likely a misconfigured or corrupt kernel, or incorrect kernel boot parameters in the **GRUB** configuration file. If the kernel has booted successfully in the past, either it has corrupted, or bad parameters were supplied. Depending on which, you can re-install the kernel, or enter into the interactive **GRUB** menu at boot and use very minimal command line parameters and try to fix that way. Or you can try booting into a rescue image.

### Kernel loads but fails to mount the root filesystem:

The main causes here are

1. Misconfigured **GRUB** configuration file.
2. Misconfigured `/etc/fstab`.
3. No support for the root filesystem type built into the kernel or in the **initramfs** image.

### Failure during the init process.

There are many things that can go wrong once **init** starts; look closely at the messages that are displayed before things stop. If things were working before, with a different kernel, that is a big clue. Look out for corrupted filesystems, errors in startup scripts etc. Try booting into a lower runlevel such as 3 (no graphics) or 1 (single user mode).

**For the exclusive use of LFS301 corp class taught 06 to 09 December**

## 43.6 Filesystem Corruption and Recovery

# Filesystem Corruption and Recovery

- **fsck** may be used to fix corrupted filesystems
- Check `/etc/fstab` for misconfiguration
- May need to run

```
$ mount -o remount,rw /
```

to attempt fixes
- Try manually mounting filesystems

**fsck** may be used to attempt repair. However, before doing that one should check that `/etc/fstab` has not been misconfigured or corrupted. Note once again you could have a problem with a filesystem type the kernel you are running does not understand.

If the root filesystem has been mounted you can examine this file, but `/` may have been mounted as read-only. so to edit the file and fix it you can run:

```
$ sudo mount -o remount,rw /
```

to remount it with write permission.

If `/etc/fstab` seems to be correct, you can move to **fsck**. First you should try:

```
$ sudo mount -a
```

to try and mount all filesystems. If this does not succeed completely, you can try to manually mount the ones with problems. You should first run **fsck** to just examine; afterwards you can run it again to have it try and fix any errors found.

## 43.7 Virtual Consoles

# Using the Virtual Consoles

- **Linux has 12 virtual consoles** by default (also called **virtual terminals**)
  - The first 6 are defined as login text consoles
  - Console 1 is used by most distributions as the system console
  - Graphical console is usually console 7, but some distributions (including **RHEL**) use console 1.
- Use Ctrl-Alt-FX to switch between consoles
  - Example: Ctrl-Alt-F5 goes to console 5.

By default, **Linux** defines 12 virtual consoles to allow local access to the system. The first 6 are usually login text consoles. The seventh is usually the graphical console, if you have one.

You can use Ctrl-Alt-FX (where X is the number of the console) to go between the console.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 43.8 Labs



### Please Note

There are no lab exercises for in this chapter. It just summarizes points discussed earlier when considering configuring and monitoring the system, and in addition, sets the stage for the following section on system rescue, which has several labs.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 44

# System Rescue



|      |                                            |     |
|------|--------------------------------------------|-----|
| 44.1 | Rescue Media and Troubleshooting . . . . . | 566 |
| 44.2 | Using Rescue/Recovery Media . . . . .      | 567 |
| 44.3 | System Rescue and Recovery . . . . .       | 568 |
| 44.4 | Emergency Boot Media . . . . .             | 569 |
| 44.5 | Using Rescue Media . . . . .               | 570 |
| 44.6 | Emergency Mode . . . . .                   | 572 |
| 44.7 | Single User Mode . . . . .                 | 573 |
| 44.8 | Labs . . . . .                             | 574 |

## 44.1 Rescue Media and Troubleshooting

# Rescue Media and Troubleshooting

- Rescue/Recover Media can also be used for troubleshooting
- Come in optical media (**CD**, **DVD**) and **USB** drives.
- Contain useful utilities for troubleshooting (and repair):
  - Disk Maintenance Utilities
  - Networking Utilities
  - Miscellaneous Utilities
  - Logging files

We will discuss system rescue and recovery shortly, but here let us focus on the use of rescue media in troubleshooting.

Exact tool choices will vary from one **Linux** distribution to another, but when you boot from an install or live **CD/DVD** or **USB** drive you will be able to select an option with a name like **Rescue Installed System**.

Rescue disks contain many useful programs, including:

- Disk utilities for creating partitions, managing **RAID** devices, managing logical volume, and creating filesystems:  
**fdisk**, **mdadm**, **pvcreate**, **vgcreate**, **lvcreate**, **mkfs**, and many others.
- Networking utilities for network debugging and network connectivity:  
**ifconfig**, **route**, **traceroute**, **mtr**, **host**, **ftp**, **scp**, **ssh**.
- Numerous other commands are also available:  
**bash**, **chroot**, **ps**, **kill**, **vi**, **dd**, **tar**, **cpio**, **gzip**, **rpm**, **mkdir**, **ls**, **cp**, **mv**, and **rm** to name a few.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 44.2 Using Rescue/Recovery Media

# Using Rescue/Recovery Media

- Will offer to mount your filesystems if possible, usually at `/mnt/sysimage`
- May offer to **chroot** into that environment.
- May offer to help install software, either from local disk or network

The rescue image will ask a number of questions upon starting. One of these is whether or not to mount your filesystems (if it can).

If so, they are mounted at somewhere, usually at `/mnt/sysimage`. You can move to that directory to get to your files or you can change into that environment with:

```
$ sudo chroot /mnt/sysimage
```

For a network-based rescue you may also be asked to mount `/mnt/source`.

You may install software packages from inside **chroot**-ed environment. You may also be able to install them from outside the chrooted environment. For example, on an **RPM**-based system, by using the `--root` option to specify the location of the root directory:

```
$ sudo rpm -ivh --force --root=/mnt/sysimage /mnt/source/Packages/vsftpd-2*.rpm
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 44.3 System Rescue and Recovery

# System Rescue and Recovery

- System rescue media come in multiple forms
- They enable you to enter **emergency mode** and perform useful operations
- It is also useful to know how to enter **single user mode**, which differs from **emergency mode**.

Sooner or later a system is likely to undergo a significant failure, such as failing to boot properly, to mount filesystems, initiate a desktop display, etc. **System Rescue** media in the form of optical disks or portable **USB** drives can be used to fix the situation. Booting into either **emergency** or **single user** mode can enable using the full suite of **Linux** tools to repair the system back to normal function.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 44.4 Emergency Boot Media

# Emergency Boot Media

- CD
- DVD
- USB
- Can be stand-alone rescue/recovery tool, a distribution installation disk, and/or a **Live Linux Distribution** with rescue/recovery capabilities

Emergency boot media are useful when your system will not boot due to some issue such as missing, misconfigured, or corrupted files or a misconfigured service.

Rescue media may also be useful if the root password is somehow lost or scrambled and needs to be reset.

Most **Linux** distributions permit the install media (**CD**, **DVD**, **USB**) and/or **Live** media to serve a double purpose as a rescue disk, which is very convenient. There are also special-purpose rescue disks available.

Live media (in any format) provide a complete and bootable operating system which runs in memory, rather than loading from disk. Users can experience and evaluate an operating system and/or **Linux** distribution without actually installing it, or making any changes to the existing operating system on the computer.

Live removable media are unique in that they can run on a computer lacking secondary storage, such as a hard disk drive, or with a corrupted hard disk drive or filesystem, allowing users to rescue data.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 44.5 Using Rescue Media

# Using Rescue Media

- Virtually all **Linux** distributions have a rescue image as a choice on the **GRUB** menu. If your system has gotten this far and the rescue option works, you should try it before mounting rescue media.
- When system boots you will either have to pick a menu choice for rescue or type something like

`Linux rescue`

at the boot: prompt

- You may be asked questions about language and other choices, such as whether to initiate the network.
- You will probably also be asked which filesystems to mount etc.

Whether you are using **Live**, install or rescue media, the procedures for entering into a special operating system for rescue and recovery are the same, and as we have pointed out one medium serves all three purposes.

The rescue/recovery mode can be accessed from an option on the boot menu when the system starts from the removable media. In many cases you may have to type `rescue` on a line like:

`boot: Linux rescue`

We cannot tell you all the possibilities as each distribution has somewhat different, but easy to ascertain procedures.

Next, you can expect to be asked some questions such as which language to continue in, as well as make some distribution-dependent choices. Then you will be prompted to select where a valid rescue image is located: **CD/DVD**, **Hard Drive**, **NFS**, **FTP**, or **HTTP**.

The selected location must contain a valid installation tree, and the installation tree must be for the same **Linux** version as the rescue disk, and if you are using removable media the installation tree must be the same as the one from which the media was created. If you are using a `boot.iso` image downloaded from the vendor, then you will also need a network-based install tree.

Additional questions are asked about mounting your filesystems. If they can be found, they are mounted under `/mnt/sysimage`. You will then be given a shell prompt and access to various utilities to make the appropriate fixes to your system.

`chroot` can be used to better access your root (`/`) filesystem.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Rescue USB Key

- **USB** install/live/rescue disk images are made available by all modern distributions.
- In most cases can be transferred to **USB** drive with:  
`$ dd if=boot.iso of=/dev/sdX`  
if `/dev/sdX` is your removable drive
- Make sure your **BIOS** is set to boot off a **USB** drive.
- Tools such as **livecd-tools** and **liveusb-creator** can make this process much easier and can also customize the disk.

Many distributions provide a `boot.iso` image file for download (the name may differ). You can use `dd` to place this on a **USB** key drive as in:

```
$ dd if=boot.iso of=/dev/sdX
```

assuming your system recognizes the removable drive as `/dev/sdX`. Be aware this will obliterate the previously existing contents on the drive!

Assuming your system has the capability of booting from **USB** media and the **BIOS** is configured for it, you can then boot from this **USB** drive. It will then function in the same fashion as a rescue **CD** or **DVD**. However, note the install tree will not be present on the **USB** drive; therefore this method requires a network-based install tree if one is needed.

Helpful utilities such as **livecd-tools** and **liveusb-creator** allow specification of either a local drive or the Internet as the location for obtaining an install image, and then do all the hard work of constructing a bootable image and burning it on the removable drive. This is extremely convenient and works for virtually all **Linux** distributions.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 44.6 Emergency Mode

# Emergency Mode

- **emergency mode** boots into the most minimal environment.
- Root filesystem in read-only
- **init** scripts not run, very little setup done
- Useful when **init** is failing
- Filesystems are still mounted, helps to recover data
- Must interact with **GRUB** interactive menu to select emergency mode.
- Will be asked for root password before getting a command prompt
- System will sometimes automatically enter emergency mode during a failed boot, like due to corrupted filesystem

In **emergency mode** you are booted into the most minimal environment possible. The root filesystem is mounted read-only, no init scripts are run and almost nothing is set up.

The main advantage of emergency mode over single-user mode (to be described next) is that if **init** is corrupted or not working, you can still mount filesystems to recover data that might be lost during a re-installation.

To enter emergency mode you need to select an entry from the **GRUB** boot menu and then hit **e** for edit. Then add the word **emergency** to the kernel command line before telling the system to boot. You will be asked for the root password before getting a shell prompt.

Note you may also enter emergency mode when a boot fails for a variety of reasons, including a corrupted filesystem.

For the exclusive use of LFS301 corp class taught 06 to 09 December

## 44.7 Single User Mode

# Single User Mode

- Used when system boots but disallows login.
- System boots to runlevel 1
  - **init** is started.
  - Services are not started.
  - Network is not activated.
  - All possible filesystems are mounted.
  - root access is granted without a password.
  - A system maintenance command line shell is launched.

If your system boots, but does not allow you to log in when it has completed booting, try **single user mode**.

In this mode your system boots to **runlevel 1** (in **SysVInit** language). Because single user mode automatically tries to mount your filesystem, you can not use it when your root filesystem cannot be mounted successfully, or if the **init** configuration is corrupted.

To boot into single user mode, you use the same method as described for emergency mode with one exception, replace the keyword **emergency** with the keyword **single**.

## 44.8 Labs

### Exercise 44.1: Preparing to use Rescue/Recover Media



#### Very Important

In the following exercises we are going to deliberately damage the system and then recover through the use of rescue media. Thus, it is obviously prudent to make sure you can indeed boot off the rescue media before you try anything more ambitious.

So first make sure you have rescue media, either a dedicated rescue/recovery image, or an install or Live image on either an optical disk or usb drive.

Boot off it and make sure you know how to force the system to boot off the rescue media (you are likely to have to fiddle with the **BIOS** settings), and when the system boots, choose rescue mode.



#### Please Note

If you are using a virtual machine, the procedure is logically the same with two differences:

- Getting to the **BIOS** might be difficult depending on the hypervisor you use. Some of them require very rapid keystrokes, so read the documentation and make sure you know how to do it.
- You can use a physical optical disk or drive, making sure the virtual machine settings have it mounted, and if it is **USB** you may have some other hurdles to make sure the virtual machine can claim the physical device. It is usually easier to simply connect a .iso image file directly to the virtual machine.

If you are working with a virtual machine, obviously things are less dangerous, and if you are afraid of corrupting the system in an unfixable way, simply make a backup copy of the virtual machine image before you do these exercises, you can always replace the image with it later.



#### Very Important

**Do not do the following exercises unless you are sure you can boot your system off rescue/recovery media!**

### Exercise 44.2: Recovering from a Corrupted GRUB Configuration

1. Edit your **GRUB** configuration file (`/boot/grub/grub.cfg`, `/boot/grub2/grub.cfg` or `/boot/grub/grub.conf`), and modify the `kernel` line by removing the first character of the value in the field named `UUID`. Take note of which character you removed, you will replace it in rescue mode. (If your root filesystem is identified by either label or hard disk device node, make an analogous simple change.) Keep a backup copy of the original.



#### On a BLSCFG System

You can corrupt the command line by editing `/etc/grub2/grubenv` instead.

2. Reboot the machine. The system will fail to boot, saying something like `No root device was found`. You will also see that a `panic` occurred.
3. Insert into your machine the **installation** or **Live DVD** or **CD** or **USB** drive (or network boot media) if you have access to a functioning installation server). Reboot again. When the boot menu appears, choose to enter rescue mode.
4. As an alternative, you can try selecting a rescue image from the **GRUB** menu; most distributions offer this. You'll get the  
**For the exclusive use of LFS301 corp class taught 06 to 09 December**

same experience as using rescue media, but it will not always work. For example, if the root filesystem is damaged it will be impossible to do anything.

5. In rescue mode, agree when asked to search for filesystems. If prompted, open a shell, and explore the rescue system by running utilities such as **mount** and **ps**.
6. Repair your broken system by fixing your **GRUB** configuration file, either by editing it or restoring from a backup copy.
7. Type **exit** to return to the installer, remove the boot media, and follow the instructions on how to reboot. Reboot your machine. It should come up normally.

### Exercise 44.3: Recovering from Password Failure

1. As root (not with **sudo**), change the root password. We will pretend we don't know what the new password is.
2. Log out and try to login again as root using the old password. Obviously you will fail.
3. Boot using the rescue media, and select Rescue when given the option. Let it mount filesystems and then go to a command line shell.
4. Go into your **chroot**-ed environment (so you have normal access to your systems):

```
$ chroot /mnt/sysimage
```

and reset the root password back to its original value.

5. Exit, remove the rescue media, and reboot, you should be able to login normally now.

### Exercise 44.4: Recovering from Partition Table Corruption



#### Very Important

This exercise is dangerous and could leave to an unusable system. Make sure you really understand things before doing it



#### Please Note

The following instructions for an **MBR** system. if you have **GPT** you need to use **sgdisk** with the **--backup-file** and **--load-backup** options as discussed in the partitioning chapter

1. Login as root and save your **MBR**:

```
$ dd if=/dev/sda of=/root/mbrsave bs=446 count=1
1+0 records in
2 1+0 records out
3 446 bytes (446 B) copied, 0.00976759 s, 45.7 kB/s
```

Be careful: make sure you issue the exact command above and that the file saved has the right length:

```
$ sudo ls -l /root/mbrsave
1 -rw-r--r-- 1 root root 446 Nov 12 07:54 mbrsave
```

2. Now we are going to obliterate the **MBR** with:

```
$ dd if=/dev/zero of=/dev/sda bs=446 count=1
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

```

1 1+0 records in
2 1+0 records out
3 446 bytes (446 B) copied, 0.000124091 s, 3.6 MB/s

```

3. Reboot the system; it should fail.
4. Reboot into the rescue environment and restore the **MBR**:

```
$ dd if=/mnt/sysimage/root/mbrsave of=/dev/sda bs=446 count=1
```

5. Exit from the rescue environment and reboot. The system should boot properly now.

## Exercise 44.5: Recovering Using the Install Image



### Please Note

This exercise has been specifically written for **Red Hat**-based systems. You should be able to easily construct the appropriate substitutions for other distribution families.

1. Remove the **zsh** package (if it is installed!):

```
s$ dnf remove zsh
```

or

```
$ rpm -e zsh
```

Note we have chosen a package that generally has no dependencies to simplify matters. If you choose something that does, you will have to watch your step in the below so that anything else you remove you reinstall as needed as well.

2. Boot into the rescue environment.

3. Re-install (or install) **zsh** from within the rescue environment. First, mount the install media at `/mnt/source`:

```
$ mount /dev/cdrom /mnt/source
```

Then reinstall the package:

```
$ rpm -ivh --force --root /mnt/sysimage /mnt/source/Packages/zsh*.rpm
```

The `--force` option tells **rpm** to use the source directory in determining dependency information etc. Note that if the install image is much older than your system which has had many updates the whole procedure might collapse!

4. Exit and reboot.

5. Check that **zsh** has been reinstalled:

```
$ rpm -q zsh
```

```
1 zsh-5.0.2-7.el7.x86_64
```

6. `$ zsh`

```
1
```

```
2 [coop@q7]/tmp/LFS201%
```

For the exclusive use of LFS301 corp class taught 06 to 09 December

## Chapter 45

# Closing and Evaluation Survey



45.1 Evaluation Survey ..... 577

## 45.1 Evaluation Survey

# Evaluation Survey

Thank you for taking this **Linux Foundation** course.

Your comments are important to us and we take them seriously, both to measure how well we fulfilled your needs and to help us improve future sessions.

- Please Evaluate your training using the link your instructor will provide.
- Please be sure to check the spelling of your name and use correct capitalization as this is how your name will appear on the certificate.
- This information will be used to generate your **Certificate of Completion**, which will be sent to the email address you have supplied, so make sure it is correct.



Figure 45.1: Course Survey

For the exclusive use of LFS301 corp class taught 06 to 09 December