LFS307

Linux System Administration

Version 1.4

Version 1.4

© Copyright the Linux Foundation 2023. All rights reserved.

© Copyright the Linux Foundation 2023. 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 go to https://trainingsupport.linuxfoundation.org.

Nondisclosure of Confidential Information

"Confidential Information" shall not include any of the following, even if marked confidential or proprietary: (a) information that relates to the code base of any open source or open standards project (collectively, "Open Project"), including any existing or future contribution thereto; (b) information generally relating or pertaining to the formation or operation of any Open Project; or (c) information relating to general business matters involving any Open Project.

This course does not include confidential information, nor should any confidential information be divulged in class.

© Copyright the Linux Foundation 2023. All rights reserved.

Contents

1 Introduction **1**
- 1.1 Linux Foundation . . . 2
- 1.2 Linux Foundation Training . . . 4
- 1.3 Linux Foundation Certifications . . . 8
- 1.4 Linux Foundation Digital Badges . . . 11
- 1.5 Laboratory Exercises, Solutions and Resources . . . 12
- 1.6 Things Change in Linux and Open Source Projects . . . 14
- 1.7 E-Learning Course: LF207 . . . 15
- 1.8 Distribution Details . . . 16
- 1.9 Labs . . . 23

2 Linux Filesystem Tree Layout **25**
- 2.1 One Big Filesystem . . . 26
- 2.2 Data Distinctions . . . 27
- 2.3 FHS Linux Standard Directory Tree . . . 28
- 2.4 root (/) directory . . . 31
- 2.5 /bin . . . 32
- 2.6 /boot . . . 34
- 2.7 /dev . . . 35
- 2.8 /etc . . . 36
- 2.9 /home . . . 37
- 2.10 /lib and /lib64 . . . 39
- 2.11 /media . . . 40
- 2.12 /mnt . . . 41
- 2.13 /opt . . . 42
- 2.14 /proc . . . 43
- 2.15 /sys . . . 45
- 2.16 /root . . . 46
- 2.17 /sbin . . . 47
- 2.18 /srv . . . 49
- 2.19 /tmp . . . 50
- 2.20 /usr . . . 51
- 2.21 /var . . . 52
- 2.22 /run . . . 53
- 2.23 Labs . . . 54

3 User Environment **57**
- 3.1 Environment Variables . . . 58
- 3.2 Command History . . . 62
- 3.3 Command Aliases . . . 66
- 3.4 Labs . . . 67

4 User Account Management **71**
- 4.1 User Accounts . . . 72
- 4.2 Shell Startup Files . . . 75
- 4.3 Management of User Accounts . . . 78

4.4 Locked Accounts . . . 80
4.5 Passwords . . . 81
4.6 /etc/shadow . . . 82
4.7 Password Management . . . 84
4.8 Password Aging . . . 85
4.9 The root Account . . . 86
4.10 SSH . . . 88
4.11 Labs . . . 95

5 Group Management **99**
5.1 Groups . . . 100
5.2 Group Membership . . . 101
5.3 Group Management . . . 102
5.4 User Private Groups . . . 103
5.5 Labs . . . 104

6 File Permissions and Ownership **107**
6.1 File Permissions and Ownership . . . 108
6.2 File Access Rights . . . 109
6.3 chmod, chown and chgrp . . . 110
6.4 umask . . . 113
6.5 Filesystem ACLs . . . 115
6.6 Labs . . . 116

7 Package Management Systems **119**
7.1 Why Use Packages? . . . 120
7.2 Software Packaging Concepts . . . 121
7.3 Package Types . . . 122
7.4 Available Package Management Systems . . . 123
7.5 Packaging Tool Levels and Varieties . . . 124
7.6 Package Sources . . . 125
7.7 Creating Software Packages . . . 126
7.8 Revision Control Systems . . . 127
7.9 Available Source Control Systems . . . 128
7.10 The Linux Kernel and git . . . 129
7.11 Labs . . . 131

8 dpkg **135**
8.1 DPKG (Debian Package) . . . 136
8.2 Package File Names and Source . . . 137
8.3 DPKG Queries . . . 138
8.4 Installing/Upgrading/Uninstalling . . . 139
8.5 Labs . . . 140

9 APT **141**
9.1 APT . . . 142
9.2 APT Utilities . . . 143
9.3 Queries . . . 144
9.4 Installing/Removing/Upgrading Packages . . . 145
9.5 Cleaning Up . . . 146
9.6 Labs . . . 147

10 RPM **149**
10.1 RPM (Red Hat Package Manager . . . 150
10.2 Package File Names . . . 151
10.3 RPM Database and Helper Programs . . . 152
10.4 Queries . . . 154
10.5 Verifying Packages . . . 155
10.6 Installing and Removing Packages . . . 157

© Copyright the Linux Foundation 2023. All rights reserved.

10.7 Updating, Upgrading and Freshening RPM Packages . . . 159
10.8 Upgrading the Linux Kernel . . . 161
10.9 rpm2archive and rpm2cpio . . . 162
10.10 Labs . . . 163

11 dnf and yum **167**
11.1 dnf . . . 168
11.2 yum . . . 169
11.3 Queries . . . 170
11.4 Installing/Removing/Upgrading Packages . . . 171
11.5 Additional dnf Commands . . . 172
11.6 Labs . . . 173

12 zypper **177**
12.1 zypper . . . 178
12.2 Queries . . . 179
12.3 Installing/Removing/Upgrading Packages . . . 180
12.4 Additional zypper Commands . . . 181
12.5 Labs . . . 182

13 GIT Fundamentals **185**
13.1 Revision Control . . . 186
13.2 Basic Commands . . . 188
13.3 Try some git commands . . . 190
13.4 Get Software with Git . . . 196
13.5 Labs . . . 202

14 Processes **205**
14.1 Programs and Processes . . . 206
14.2 Process Limits . . . 210
14.3 Creating Processes . . . 213
14.4 Process Control . . . 215
14.5 Starting Processes in the Future . . . 218
14.6 Process States . . . 222
14.7 Execution Modes . . . 223
14.8 Daemons . . . 226
14.9 niceness . . . 227
14.10 Labs . . . 229

15 Process Monitoring **237**
15.1 Process Monitoring . . . 238
15.2 Troubleshooting . . . 239
15.3 ps . . . 241
15.4 pstree . . . 243
15.5 top . . . 245
15.6 Labs . . . 247

16 Memory Monitoring, Usage and Configuring Swap **249**
16.1 Memory Monitoring and Tuning . . . 250
16.2 /proc/sys/vm . . . 252
16.3 vmstat . . . 253
16.4 Swap . . . 254
16.5 Out of Memory Killer (OOM) . . . 255
16.6 Labs . . . 257

17 I/O Monitoring and Tuning **261**
17.1 I/O Monitoring . . . 262
17.2 iostat . . . 263
17.3 iotop . . . 264

© Copyright the Linux Foundation 2023. All rights reserved.

17.4 Labs . . . 265

18 Containers Overview **269**
18.1 Containers . . . 270
18.2 Containers vs Virtual Machines . . . 271
18.3 Docker . . . 272
18.4 Cowsay Example . . . 275
18.5 Reproducible Builds . . . 279
18.6 Labs . . . 283

19 Linux Filesystems and the VFS **287**
19.1 Filesystem Basics . . . 288
19.2 Filesystem Concepts . . . 289
19.3 Virtual Filesystem (VFS) . . . 291
19.4 Available Filesystems . . . 292
19.5 Journalling Filesystems . . . 294
19.6 Special Filesystems . . . 295
19.7 Labs . . . 296

20 Disk Partitioning **299**
20.1 Common Disk Types . . . 300
20.2 Disk Geometry . . . 301
20.3 Partitioning . . . 302
20.4 Partition Tables . . . 304
20.5 Naming Disk Devices . . . 306
20.6 blkid and lsblk . . . 307
20.7 Sizing up partitions . . . 309
20.8 Backing Up and Restoring Partition Tables . . . 310
20.9 Partition table editors . . . 312
20.10 fdisk . . . 313
20.11 Labs . . . 314

21 Filesystem Features: Attributes, Creating, Checking, Usage, Mounting **319**
21.1 Extended Attributes . . . 320
21.2 Creating and formatting filesystems . . . 321
21.3 Troubleshooting Filesystems . . . 322
21.4 Checking and Repairing Filesystems . . . 324
21.5 Filesystem Usage . . . 325
21.6 Disk Usage . . . 326
21.7 Mounting filesystems . . . 327
21.8 NFS . . . 331
21.9 Mounting at Boot and /etc/fstab . . . 332
21.10 automount . . . 334
21.11 Network Block Devices . . . 336
21.12 Labs . . . 340

22 The Ext4 Filesystems **349**
22.1 ext4 Features . . . 350
22.2 ext4 Layout and Superblock and Block Groups . . . 351
22.3 dumpe2fs . . . 354
22.4 tune2fs . . . 355
22.5 Labs . . . 356

23 Logical Volume Management (LVM) **361**
23.1 Logical Volume Management (LVM) . . . 362
23.2 Volumes and Volume Groups . . . 363
23.3 Working with Logical Volumes . . . 364
23.4 Resizing Logical Volumes . . . 367
23.5 LVM Snapshots ** . . . 368

© Copyright the Linux Foundation 2023. All rights reserved.

23.6 Labs . . . 369

24 Kernel Services and Configuration **371**
24.1 Kernel Overview . . . 372
24.2 Kernel Boot Parameters . . . 373
24.3 Kernel Command Line . . . 374
24.4 Boot Process Failures . . . 375
24.5 sysctl . . . 376
24.6 Labs . . . 377

25 Kernel Modules **379**
25.1 Kernel Modules . . . 380
25.2 Module Utilities . . . 382
25.3 modinfo . . . 384
25.4 Module Configuration . . . 385
25.5 Labs . . . 386

26 Devices and udev **387**
26.1 udev and Device Management . . . 388
26.2 Device Nodes . . . 389
26.3 Rules . . . 392
26.4 Labs . . . 395

27 Network Addresses **397**
27.1 IP Addresses . . . 398
27.2 IPv4 Address Types . . . 399
27.3 IPv6 Address Types . . . 401
27.4 IP Address Classes . . . 402
27.5 Netmasks . . . 403
27.6 Hostnames . . . 404
27.7 NTP . . . 405
27.8 Labs . . . 411

28 Network Devices and Configuration **413**
28.1 Network Devices . . . 414
28.2 Predictable Network Interface Device Names . . . 415
28.3 Network Configuration Files . . . 417
28.4 Network Manager . . . 419
28.5 Routing . . . 424
28.6 Virtual Network Interfaces . . . 427
28.7 DNS and Name Resolution . . . 431
28.8 Network Troubleshooting . . . 436
28.9 Network Diagnostics . . . 437
28.10 Labs . . . 441

29 LDAP ** **445**
29.1 LDAP Authentication . . . 446
29.2 Labs ** . . . 450

30 Firewalls **459**
30.1 Firewalls . . . 460
30.2 Interfaces . . . 463
30.3 firewalld . . . 465
30.4 Zones . . . 468
30.5 Source Management . . . 472
30.6 Service and Port Management . . . 473
30.8 Labs . . . 481

31 System Init: systemd history and customization **485**

© Copyright the Linux Foundation 2023. All rights reserved.

31.1 The init Process . . . 486
31.2 Startup Alternatives . . . 487
31.3 systemd . . . 488
31.4 systemctl . . . 490
31.5 Labs . . . 491

32 Backup and Recovery Methods **495**
32.1 Backup Basics . . . 496
32.2 Backup vs Archive . . . 498
32.3 Backup Methods and Strategies . . . 500
32.4 tar . . . 503
32.5 Compression: gzip, bzip2 and xz and Backups . . . 506
32.6 dd . . . 507
32.7 rsync . . . 508
32.8 Backup Programs ** . . . 509
32.9 Labs . . . 510

33 Linux Security Modules **513**
33.1 Linux Security Modules . . . 514
33.2 SELinux . . . 516
33.3 AppArmor . . . 531
33.4 Labs . . . 537

34 System Rescue **543**
34.1 Rescue Media and Troubleshooting . . . 544
34.2 Using Rescue/Recovery Media . . . 545
34.3 System Rescue and Recovery . . . 546
34.4 Emergency Boot Media . . . 547
34.5 Using Rescue Media . . . 548
34.6 Emergency Mode . . . 550
34.7 Single User Mode . . . 551
34.8 Labs . . . 552

35 Closing and Evaluation Survey **555**
35.1 Evaluation Survey . . . 555

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.

© Copyright the Linux Foundation 2023. All rights reserved.

List of Figures

2.1 /bin Directory: Ubuntu 18.04 . . . 33
2.2 /boot Directory on Ubuntu 20.04 . . . 34
2.3 Device Nodes . . . 35
2.4 /dev Directory . . . 35
2.5 /home Directory . . . 38
2.6 /proc Directory . . . 44
2.7 /proc/[pid] Directory . . . 44
2.8 Contents of /proc/interrupts . . . 44
2.9 /sys Directory . . . 45
2.10 /root Directory . . . 46
2.11 /sbin Directory: Ubuntu 18.04 . . . 48
2.12 /var Directory . . . 52
2.13 /run Directory . . . 53

4.1 Using usermod . . . 79
4.2 Using chage . . . 85

6.1 Using chmod . . . 111

7.1 RPM and APT . . . 123

8.1 Getting Source Packages on Ubuntu . . . 137

10.1 Uninstalling RPM Packages . . . 158

11.1 rpm Repositories . . . 168

14.1 ulimit . . . 210
14.2 User and Kernel Modes and System Calls . . . 224

15.1 Using ps With BSD Options . . . 241
15.2 Customizing ps Output . . . 242
15.3 Using ps With UNIX Options . . . 242
15.4 Using top . . . 245
15.5 /proc Contents . . . 246

16.1 /proc/sys/vm . . . 252
16.2 vmstat . . . 253
16.3 Using vmstat . . . 253
16.4 Using vmstat on One Disk . . . 253

17.1 iostat . . . 263
17.2 Using iostat . . . 263
17.3 Using iotop . . . 264
17.4 iotop options . . . 264

18.1 Linux Containers vs OCI Containers . . . 270

19.1 Hard and Soft Links . . . 290

20.1 Using fdisk . . . 301
20.2 MBR Disk Partition Table . . . 304
20.3 GPT Disk Partition Table . . . 305
20.4 Using blkid . . . 307
20.5 Using lsblk . . . 308

21.1 mkfs . . . 321
21.2 fsck . . . 324
21.3 Using df . . . 325
21.4 Using du . . . 326
21.5 mount . . . 328
21.6 Currently Mounted Filesystems . . . 329
21.7 /etc/fstab Example . . . 333
21.8 automount Example . . . 335
21.9 Network Block Device . . . 339

22.1 ext[3,4] Filesystem Layout . . . 351
22.2 Block Groups . . . 352

23.1 LVM components . . . 363
23.2 Logical Volume Utilities . . . 364
23.3 Physical Volume Utilities . . . 364
23.4 Volume Group Utilities . . . 366

24.1 sysctl . . . 376

25.1 Using lsmod . . . 381
25.2 modinfo . . . 384

26.1 Device Nodes . . . 390
26.2 udev Rules . . . 391

27.1 IP Addresses . . . 398
27.2 Network Time Protocol . . . 405

28.1 **nmtui** Main Screen . . . 421
28.2 **nmtui** Edit Screen . . . 421
28.3 **nmtui** Wireless Configuration . . . 422
28.4 Using route and ip route . . . 424
28.5 DNS . . . 434
28.6 ping . . . 438
28.7 traceroute . . . 439
28.8 mtr . . . 440

29.1 LDAP Client Authentication Process . . . 448
29.2 Turnkey OpenLDAP Appliance Services . . . 450
29.3 Wireshark LDAP packet trace . . . 456

30.1 Destination NAT . . . 478
30.2 Source NAT and MASQUERADE . . . 480
30.3 SNAT or MASQUERADE example . . . 483

33.1 SELinux Enforcing Mode . . . 517
33.2 SELinux Permissive Mode . . . 517
33.3 semanage . . . 529

35.1 Course Survey . . . 555

© Copyright the Linux Foundation 2023. All rights reserved.

List of Tables

2.1 Main Directories Part I . . . 29
2.2 Main Directories Part II . . . 30
2.3 Directories Under `/usr` . . . 51
2.4 Directories Under `/var` . . . 52

7.1 Available Source Control Systems . . . 128

10.1 **rpm** Query Command Examples . . . 154

15.1 Process and Load Monitoring Tools . . . 238

16.1 Memory Monitoring Utilities . . . 250

19.1 Special Filesystems . . . 295

33.1 AppArmor Utilities . . . 535

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 1

Introduction

1.1 Linux Foundation 2
1.2 Linux Foundation Training 4
1.3 Linux Foundation Certifications 8
1.4 Linux Foundation Digital Badges 11
1.5 Laboratory Exercises, Solutions and Resources 12
1.6 Things Change in Linux and Open Source Projects 14
1.7 E-Learning Course: LF207 15
1.8 Distribution Details 16
1.9 Labs 23

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://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.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

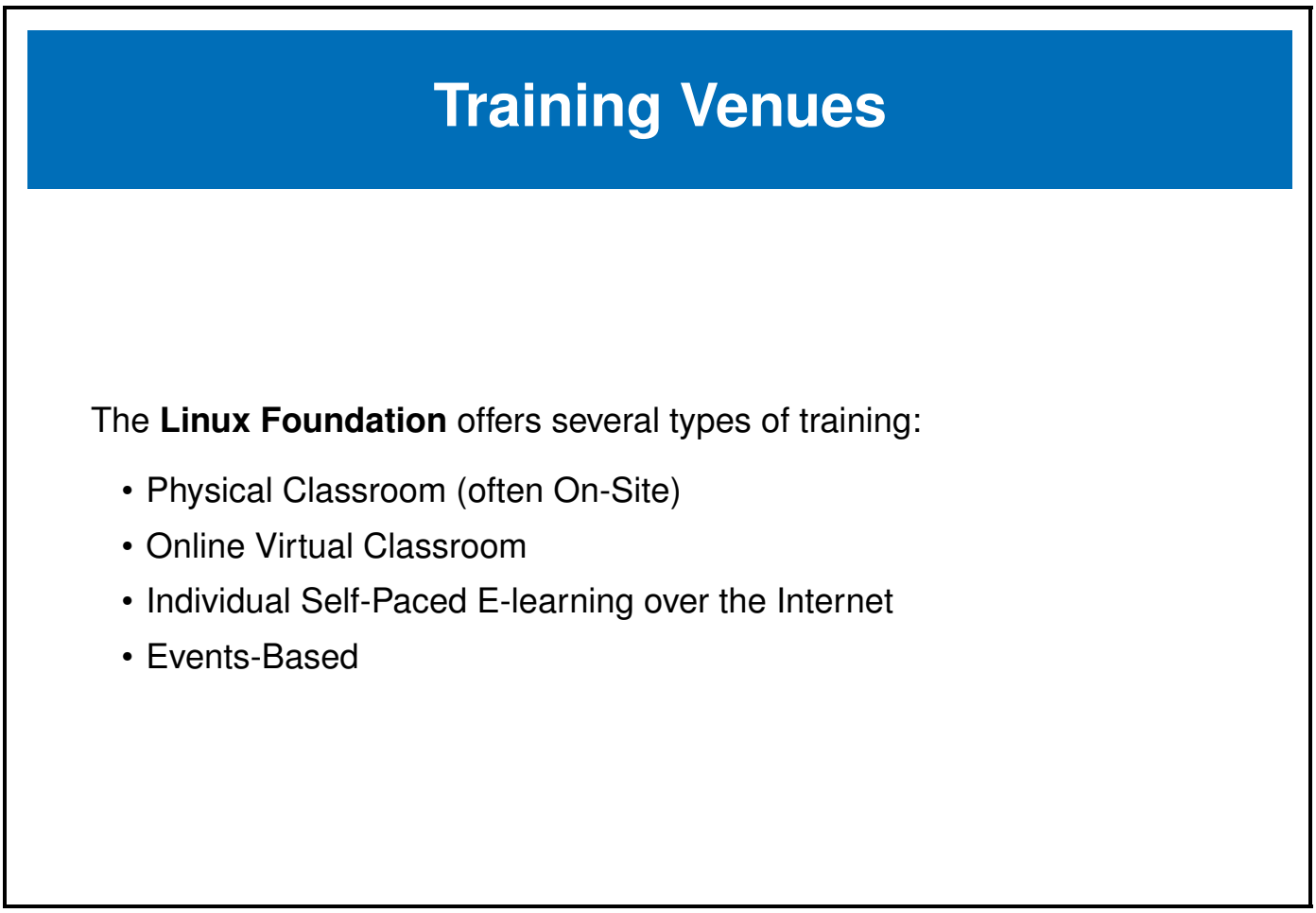
- Open Source Summit
- Embedded Linux Conference North
- Open Networking & Edge Summit
- KubeCon + CloudNativeCon
- Automotive Linux Summit
- KVM Forum
- Linux Storage Filesystem and Memory Management Summit
- Linux Security Summit
- Linux Kernel Maintainer Summit
- The Linux Foundation Member Summit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

© Copyright the Linux Foundation 2023. All rights reserved.

1.2 Linux Foundation Training



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.

© Copyright the Linux Foundation 2023. All rights reserved.

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 **MOOC**s (**M**assively **O**pen **O**nline **C**ourses) offered through **edX** at https://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"`

© Copyright the Linux Foundation 2023. All rights reserved.

Copyright

- The contents of this course and all its related materials, including handouts, are © Copyright the Linux Foundation 2023. 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 go to https://trainingsupport.linuxfoundation.org.

© Copyright the Linux Foundation 2023. All rights reserved.

About Confidential Information

Nondisclosure of Confidential Information

"Confidential Information" shall not include any of the following, even if marked confidential or proprietary: (a) information that relates to the code base of any open source or open standards project (collectively, "Open Project"), including any existing or future contribution thereto; (b) information generally relating or pertaining to the formation or operation of any Open Project; or (c) information relating to general business matters involving any Open Project.

This course does not include confidential information, nor should any confidential information be divulged in class.

© Copyright the Linux Foundation 2023. All rights reserved.

1.3 Linux Foundation Certifications

Linux Foundation Certification Exams

- The **Linux Foundation** offers comprehensive **Certification Programs**.
- 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** (**L**inux **F**oundation **C**ertified **S**ysadmin) exam, the **Linux Foundation** provides certification exams for many other open source projects. The list is constantly expanding so please see https://training.linuxfoundation.org/certification for the current list.
- For additional information, including course descriptions, technical requirements and other logistics, see https://training.linuxfoundation.org.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

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

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

© Copyright the Linux Foundation 2023. All rights reserved.

1.4 Linux Foundation Digital Badges

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.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

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/LFS307
- If you do not have a browser available you can obtain with:

```
$ wget --user=LFtraining --password=<ask instructor> \
  https://training.linuxfoundation.org/cm/LFS307/LFS307_V1.4_SOLUTIONS.tar.xz
```

  and similarly for the RESOURCES file.

```
$ wget --user=LFtraining --password=<ask instructor> \
  https://training.linuxfoundation.org/cm/LFS307/LFS307_V1.4_RESOURCES.tar
```

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

```
$ tar xvf LFS307_V1.4_SOLUTIONS.tar.xz
$ tar xvf LFS307_V1.4_RESOURCES.tar
```

You will see subdirectories in the 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.

© Copyright the Linux Foundation 2023. All rights reserved.

1.6 Things Change in Linux and Open Source Projects

Open Source Software Changes Often

- **Linux Foundation** courses are constantly updated to synchronize with upstream versions of projects such as **Kubernetes** or the **Linux** kernel.
- There will be unavoidable breakage from time to time and deprecated features will disappear.
- We try to minimize problems, but often we have to respond to upstream changes we cannot control.
- The alternative is to have stale material which is lacking important features.
- Working with these shifting sources and targets is part of the *fun* of working with open source software!

Linux Foundation courses are constantly updated, some of them with every new version release of projects they depend on, such as the **Linux** kernel or **Kubernetes**.

For example, no matter how hard we have worked to stay current, **Linux** is constantly evolving, both at the technical level (including kernel features) and at the distribution and interface level.

So please keep in mind we have tried to be as up to date as possible at the time this class was released, but there will be changes and new features we have not discussed. It is unavoidable.

As a result of this churn, no matter how hard we try there are inevitably features in the material that might suffer from one of the following:

- They will stop working on a **Linux** distribution we support.
- A change in **API** will break a laboratory exercise with a particular kernel or upstream library or product.
- Deprecated features we have been supporting will finally disappear.

There are students who just expect everything in the class to always work all the time. While we strive to achieve this, we know:

- It is an unrealistic expectation.
- Figuring out the problems and how to solve this is an active part of the class, part of the “adventure” of working with cutting edge open source projects. Moving targets are always harder to hit.

If we tried to avoid all such problems, we would have material that goes stale quickly and does not keep up with changes. We rely on our instructors and students to notice any such problems and inform courseware designers so they can incorporate the appropriate remedies in the next release.

This is the true open source way of doing things.

© Copyright the Linux Foundation 2023. All rights reserved.

1.7 E-Learning Course: LF207

Companion E-Learning Course: LF207

- The **Linux Foundation** offers a self-paced, e-learning closely related course:

  LF207: Linux System Administration Essentials
- This course covers much of the same material and shares many of the laboratory exercises as this course.
- Access to materials on line 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 **LF207** as part of your enrollment in this course. Otherwise you can contact separately through https://training.linuxfoundation.org/.

© Copyright the Linux Foundation 2023. All rights reserved.

1.8 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.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

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 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.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

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

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

1.9 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.

© Copyright the Linux Foundation 2023. All rights reserved.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 2

Linux Filesystem Tree Layout

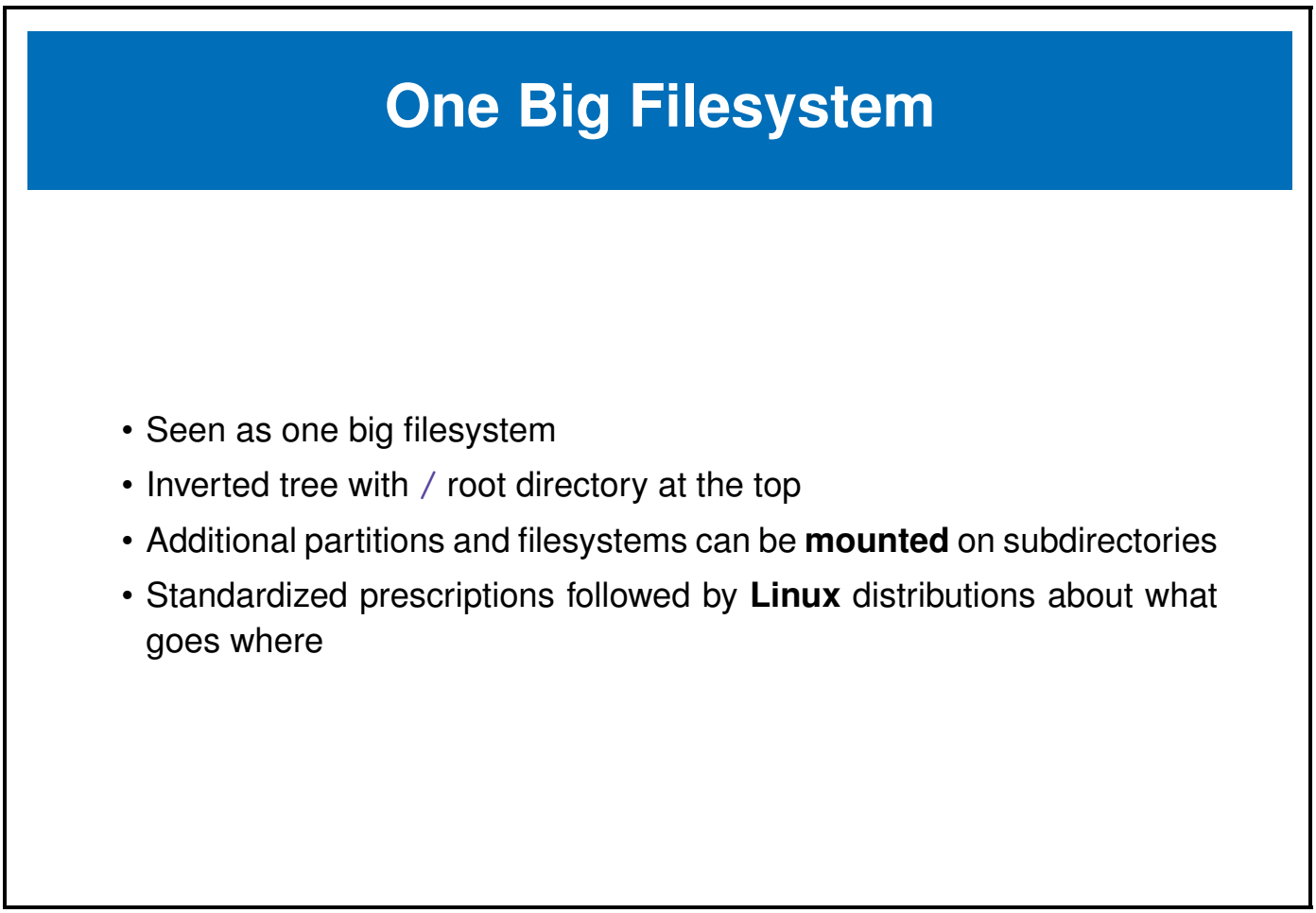
2.1 One Big Filesystem . . . 26
2.2 Data Distinctions . . . 27
2.3 FHS Linux Standard Directory Tree . . . 28
2.4 root (/) directory . . . 31
2.5 /bin . . . 32
2.6 /boot . . . 34
2.7 /dev . . . 35
2.8 /etc . . . 36
2.9 /home . . . 37
2.10 /lib and /lib64 . . . 39
2.11 /media . . . 40
2.12 /mnt . . . 41
2.13 /opt . . . 42
2.14 /proc . . . 43
2.15 /sys . . . 45
2.16 /root . . . 46
2.17 /sbin . . . 47
2.18 /srv . . . 49
2.19 /tmp . . . 50
2.20 /usr . . . 51
2.21 /var . . . 52
2.22 /run . . . 53
2.23 Labs . . . 54

Learning Objectives

By the end of this session, you should be able to:

- Explain why Linux requires the organization of one big filesystem tree, and what the major considerations are for how it is done.
- Explain the role played by the Filesystem Hierarchy Standard. (**FHS**)
- Describe what must be available at boot in the root (/) directory, and what can be available only once the system has started.
- Explore each of the main subdirectory trees, explain their purposes, and examine their contents.

2.1 One Big Filesystem



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.

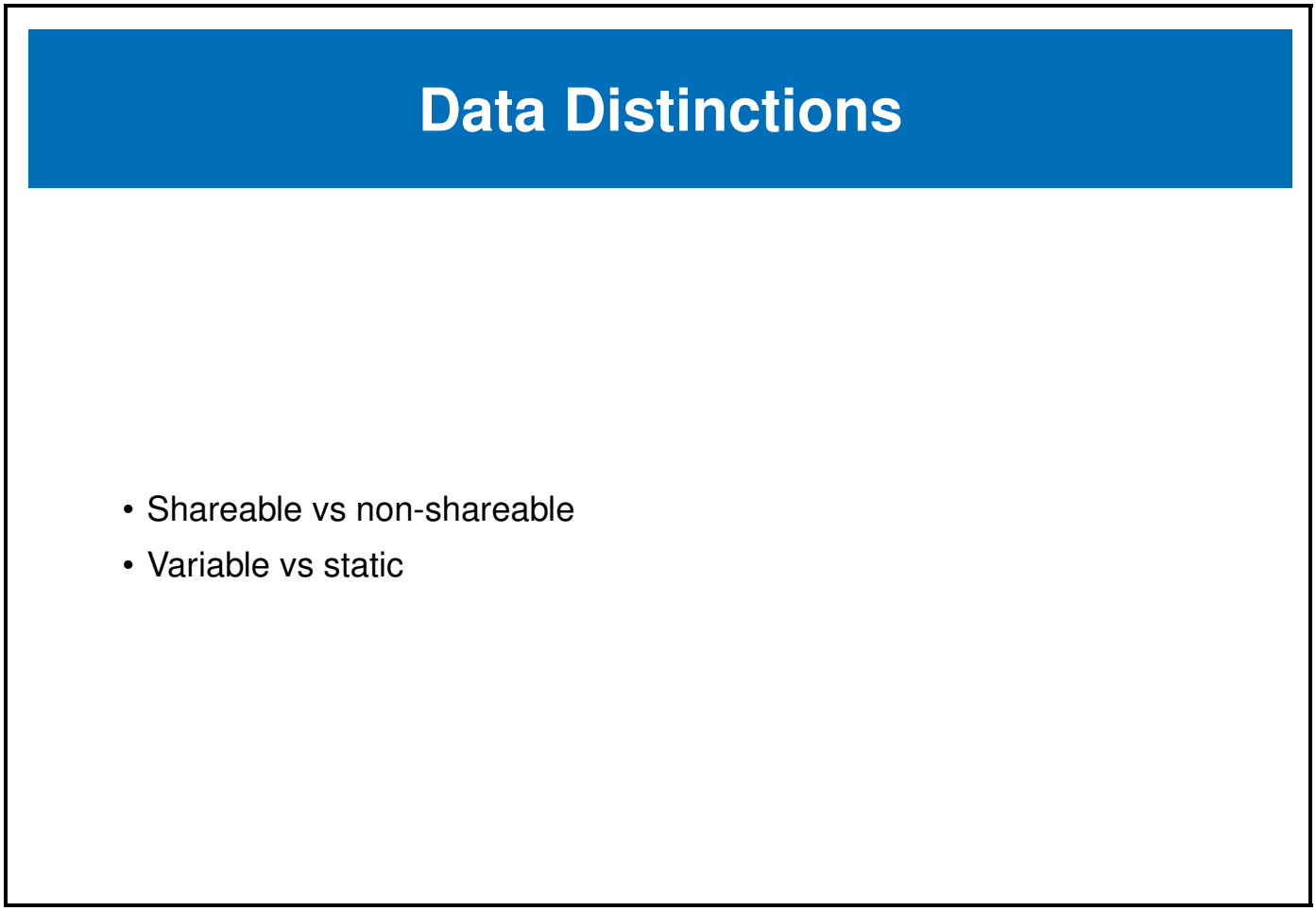
Regardless of exactly how things are joined together, it all just looks like one big filesystem; applications do not usually care at all about what physical device files actually reside on.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

2.2 Data Distinctions



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.

© Copyright the Linux Foundation 2023. All rights reserved.

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 **FSH** simplifies predictions of file locations. While most **Linux** distributions respect the **FHS**, probably none of them follow it exactly, and the last official version might not take into account some new developments.

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

© Copyright the Linux Foundation 2023. All rights reserved.

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 **CD**s, **DVD**s, **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.

© Copyright the Linux Foundation 2023. All rights reserved.

FHS Linux Standard Directory Tree, Continued

Table 2.2: **Main Directories Part II**

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

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

© Copyright the Linux Foundation 2023. All rights reserved.

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!

© Copyright the Linux Foundation 2023. All rights reserved.

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**.

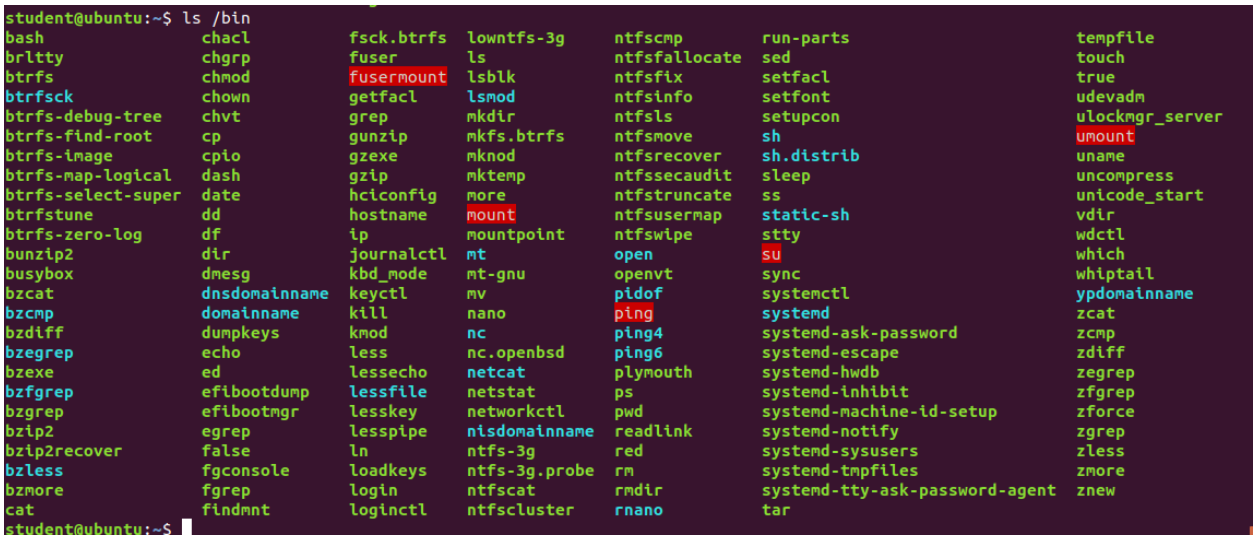
[and **test** may be there as well, and optionally, it may include:

csh, ed, tar, cpio, gunzip, 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.

© Copyright the Linux Foundation 2023. All rights reserved.

/bin Example



```
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          ntfsfix        setfacl                        true
btrfsck             chown          getfacl     lsmod          ntfsinfo       setfont                        udevadm
btrfs-debug-tree    chvt           grep        mkdir          ntfsls         setupcon                       ulockmgr_server
btrfs-find-root     cp             gunzip      mkfs.btrfs     ntfsmove       sh                             umount
btrfs-image         cpio           gzexe       mknod          ntfsrecover    sh.distrib                     uname
btrfs-map-logical   dash           gzip        mktemp         ntfssecaudit   sleep                          uncompress
btrfs-select-super  date           hciconfig   more           ntfstruncate   ss                             unicode_start
btrfstune           dd             hostname    mount          ntfsusermap    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                 zdiff
bzexe               ed             lessecho    netcat         plymouth       systemd-hwdb                   zegrep
bzfgrep             efibootdump    lessfile    netstat        ps             systemd-inhibit                zfgrep
bzgrep              efibootmgr     lesskey     networkctl     pwd            systemd-machine-id-setup       zforce
bzip2               egrep          lesspipe    nisdomainname  readlink       systemd-notify                 zgrep
bzip2recover        false          ln          ntfs-3g        red            systemd-sysusers               zless
bzless              fgconsole      loadkeys    ntfs-3g.probe  rm             systemd-tmpfiles               zmore
bzmore              fgrep          login       ntfscat        rmdir          systemd-tty-ask-password-agent znew
cat                 findmnt        loginctl    ntfscluster    rnano          tar
student@ubuntu:~$
```

Figure 2.1: **/bin Directory: Ubuntu 18.04**

Recent distribution versions of **RHEL**, **CentOS**, **Fedora**, 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.

© Copyright the Linux Foundation 2023. All rights reserved.

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:

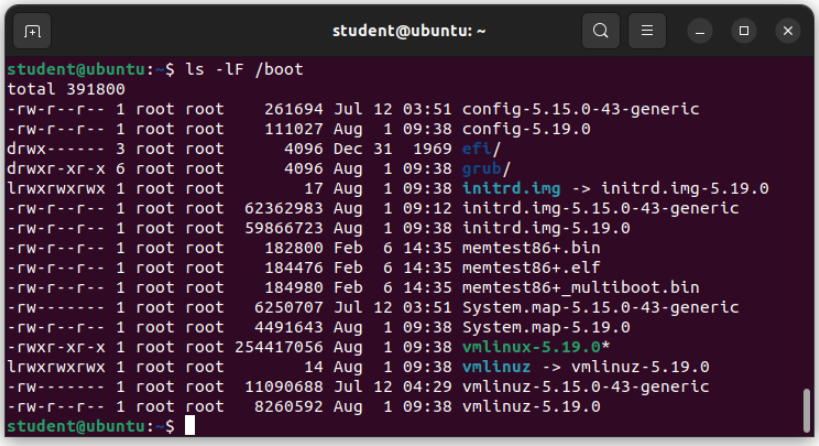


Figure 2.2: /**boot Directory on Ubuntu 20.04**

These files have longer names which depend on the **Linux** distribution and kernel version. Also, instead of `initramfs`, one might have `initrd` (**init**ial **r**am **d**isk.)

© Copyright the Linux Foundation 2023. All rights reserved.

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

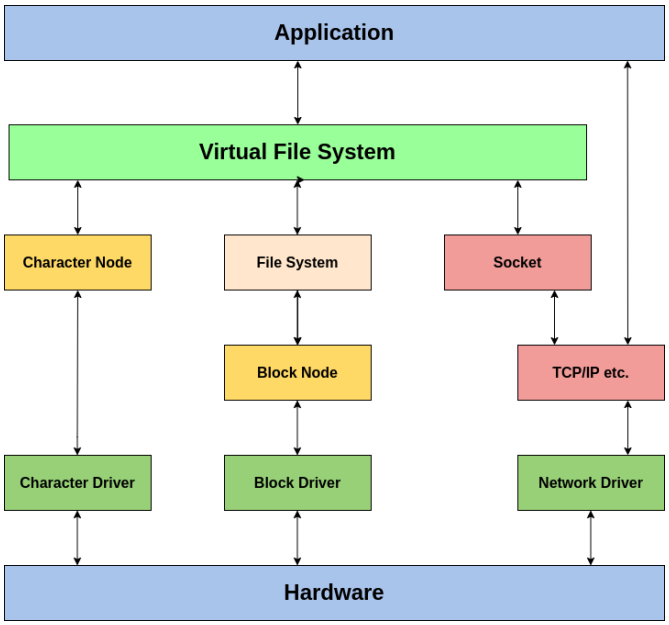


Figure 2.3: **Device Nodes**

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 devices nodes in **Linux** and are instead referenced by name, such as `eth1` or `wlan0`.

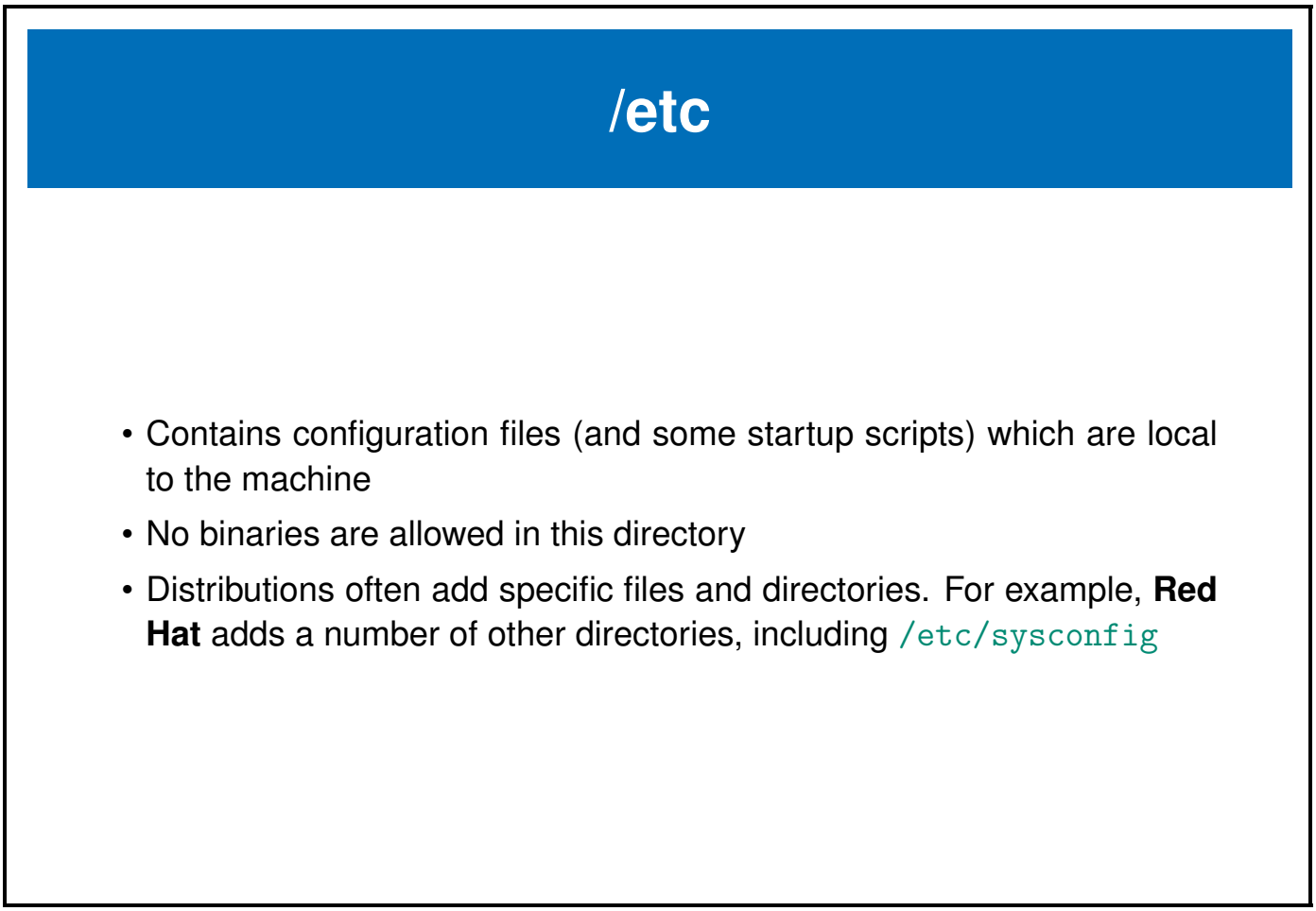
```
File  Edit  View  Search  Terminal  Help
c7:/tmp>ls -l /dev
total 0
....
crw-------   1 root root        5,   1 May 26 07:05 console
....
lrwxrwxrwx   1 root root           13 May 26 07:04 fd -> /proc/self/fd
....
brw-rw----   1 root disk        7,   0 May 26 07:05 loop0
crw-rw----   1 root disk       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--w----   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.

© Copyright the Linux Foundation 2023. All rights reserved.

2.8 /etc



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.

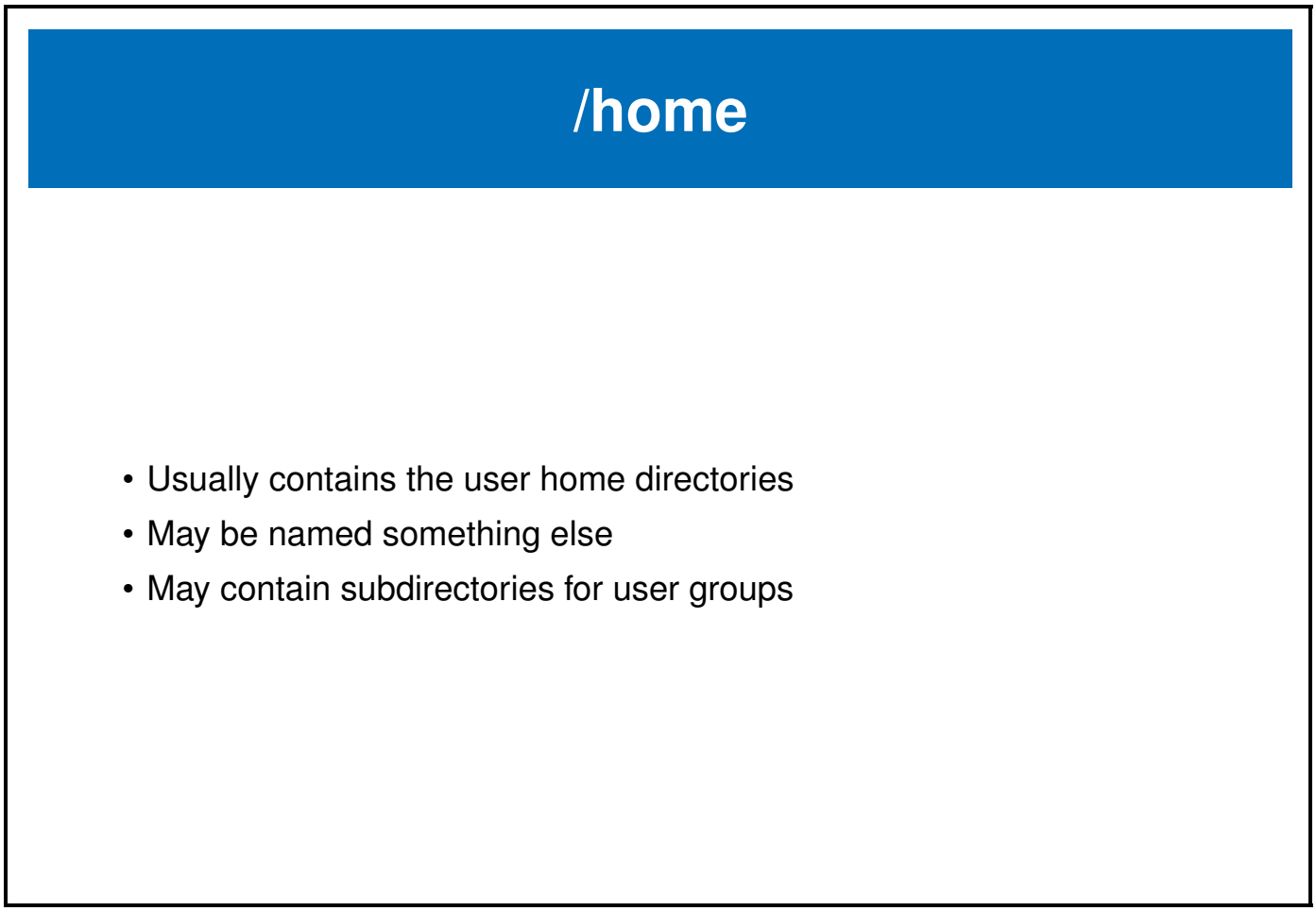
Distributions often add configuration files and directories to `/etc`. For example, **Red Hat** adds a number of other directories including `/etc/sysconfig`, where a number of system configuration files and directories live.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

2.9 /home



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.

© Copyright the Linux Foundation 2023. All rights reserved.

/home Example

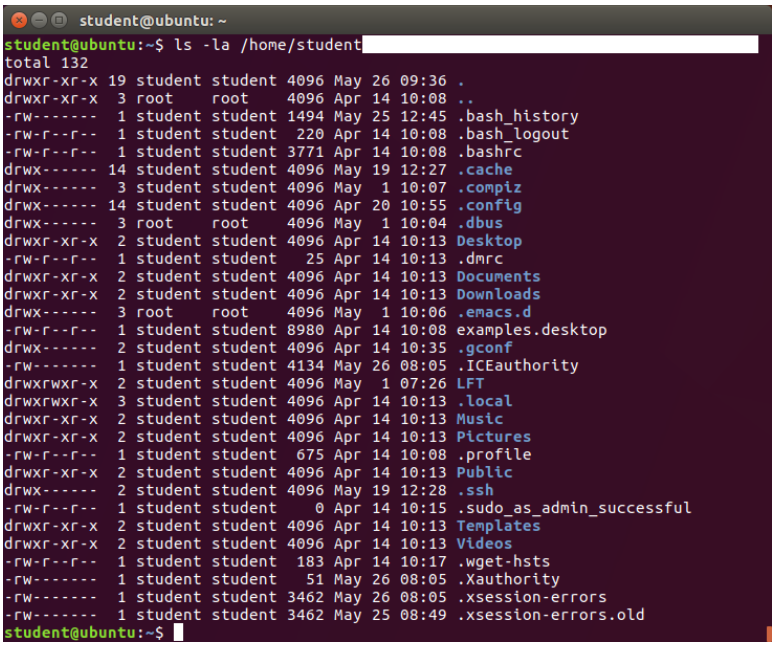


Figure 2.5: /**home Directory**

Depending on how many times you have upgraded your system or copied your **home** directory around, there can be a great variety of files:

- media and data in **XDG** specified directories such as `~/Documents`, `~/Downloads`, `~/Pictures`, and `~/Desktop`.
- program user data in **dotfiles** or in the newer locations like `~/.config` and `~/.local`
- even log files such as `~/.xsession-errors`

© Copyright the Linux Foundation 2023. All rights reserved.

2.10 /lib and /lib64

/lib and /lib64

- Contain only those libraries which are needed to execute binaries in `/bin` and `/sbin`
- Some distributions put 64-bit libraries in `/lib64`
- Kernel modules go in `/lib/modules`
- **PAM** modules go in distribution-dependent locations such as `/lib64/security` or `/lib/x86_64-linux-gnu/security`

`/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>`, as in `/lib/modules/5.19.4`.

PAM (**P**luggable **A**uthentication **M**odules) files are stored in distribution-dependent locations such as `/lib64/security` or `/lib/x86_64-linux-gnu/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*
```

```
lrwxrwxrwx 1 root root 7 Apr 23  2020 /lib -> usr/lib
lrwxrwxrwx 1 root root 9 Apr 23  2020 /lib64 -> usr/lib64
```


© Copyright the Linux Foundation 2023. All rights reserved.

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 may use /run/media/[username]/... instead of `/media`

This directory was typically used to mount filesystems on removable media. These include **CD**s, **DVD**s, 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.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

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 20K
drwxr-xr-x.  4 root root 4.0K Jul 26  2021 brother
drwxr-xr-x.  4 root root 4.0K Sep  1  2019 google
drwxr-xr-x   3 root root 4.0K Jul  7 08:03 rh
drwxr-xr-x  12 root root 4.0K Jul  1 09:55 VirtualBox
drwxr-xr-x. 30 root root 4.0K May 30 13:08 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.

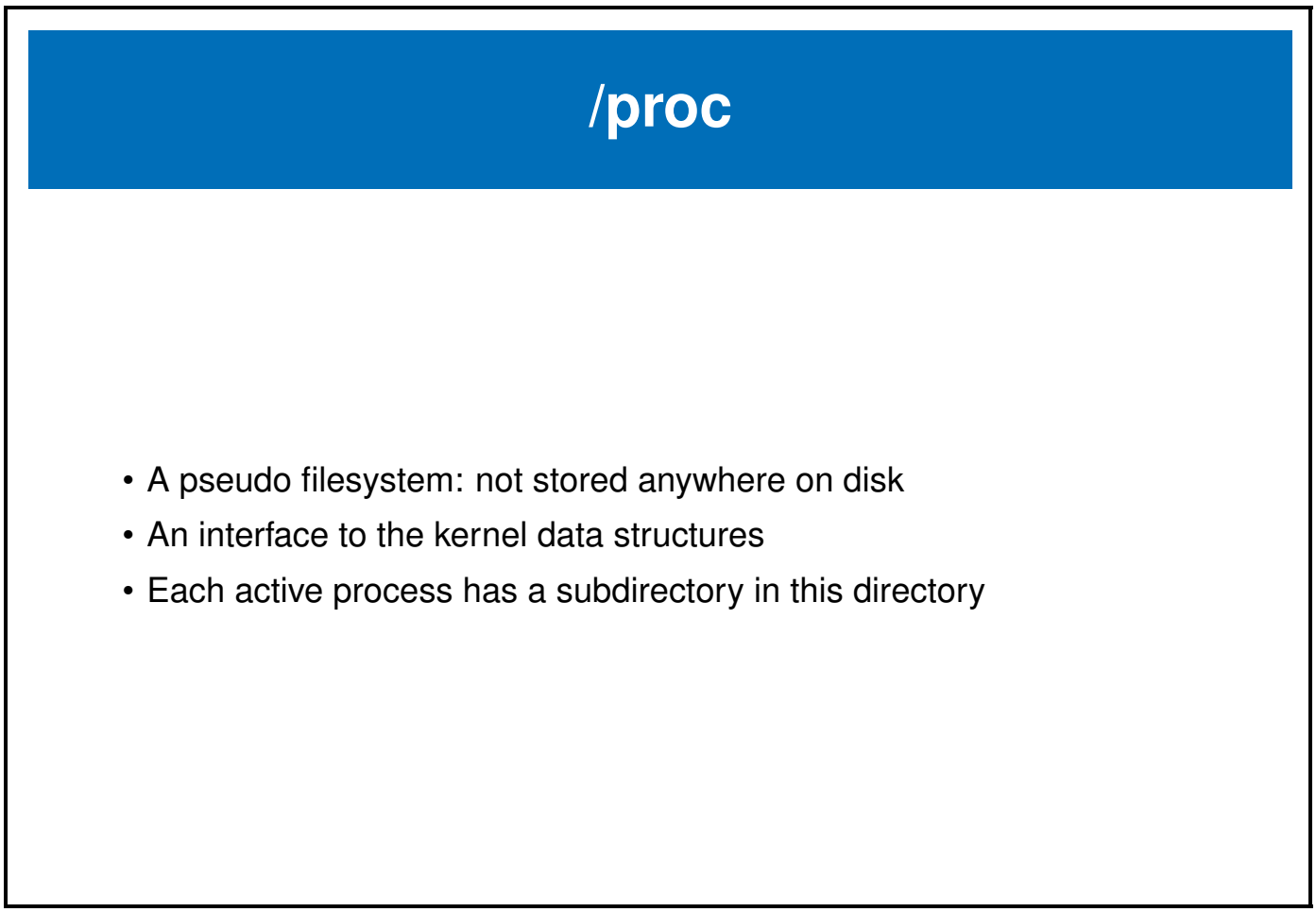
Note, however, if one uses packaging systems such as **RPM** and **APT**, as we shall discuss later, it is also easy to install and uninstall with a clear sense of file manifests and locations, without exhibiting such antisocial behavior.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

2.14 /proc



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 subdirectories. 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 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.

© Copyright the Linux Foundation 2023. All rights reserved.

/proc Example

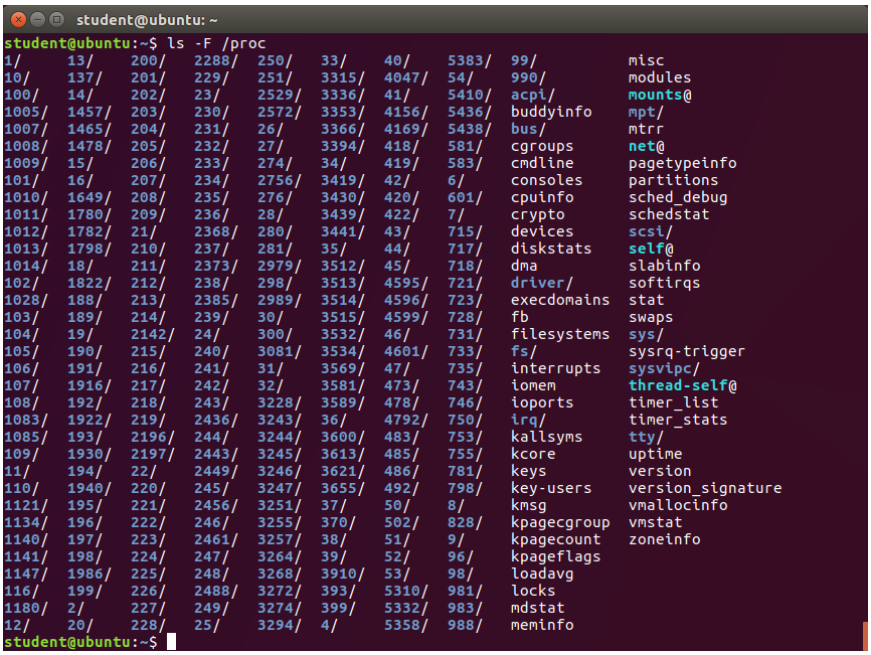


Figure 2.6: /**proc Directory**

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

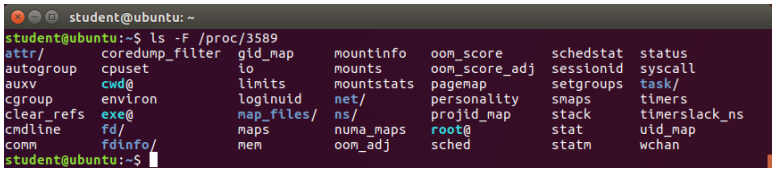


Figure 2.7: /**proc/[pid] Directory**

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

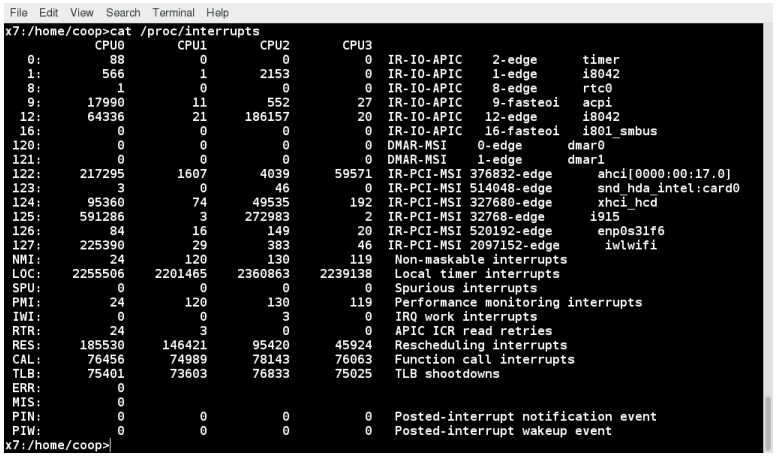


Figure 2.8: **Contents of /proc/interrupts**

© Copyright the Linux Foundation 2023. All rights reserved.

2.15 /sys

/sys

- Another pseudo filesystem: 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.

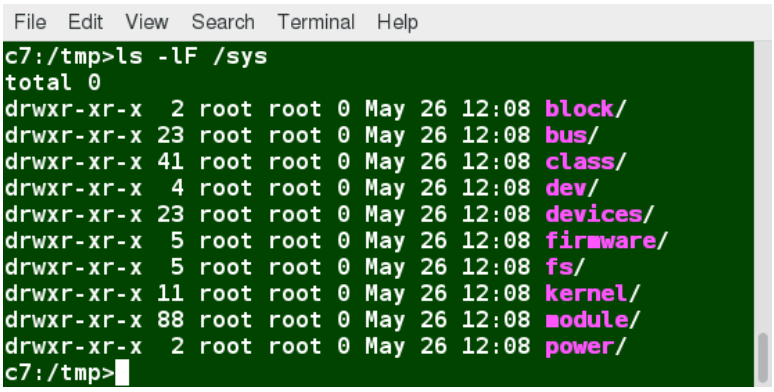


Figure 2.9: /**sys Directory**

© Copyright the Linux Foundation 2023. All rights reserved.

2.16 /root

/root

- Used as the home directory for the root user

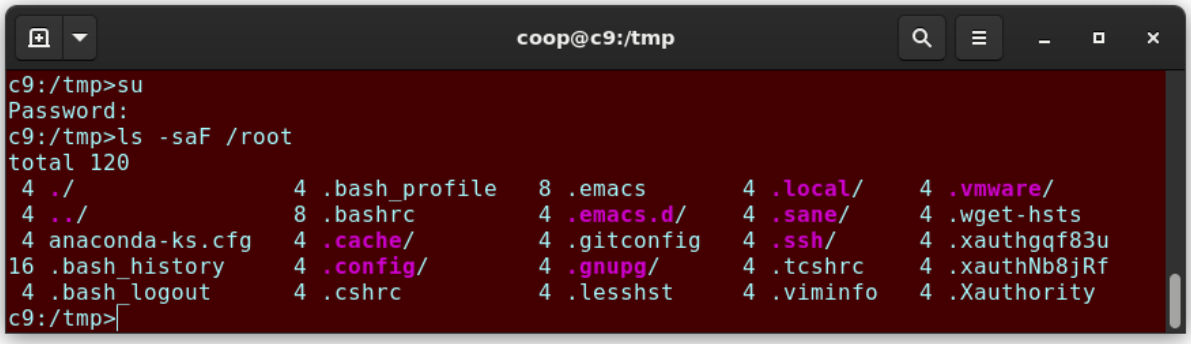


Figure 2.10: /**root Directory**

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.

© Copyright the Linux Foundation 2023. All rights reserved.

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
```

```
rwxrwxrwx 1 root root 8 Apr 23 08:23 /sbin -> usr/sbin
```


© Copyright the Linux Foundation 2023. All rights reserved.

/sbin Example

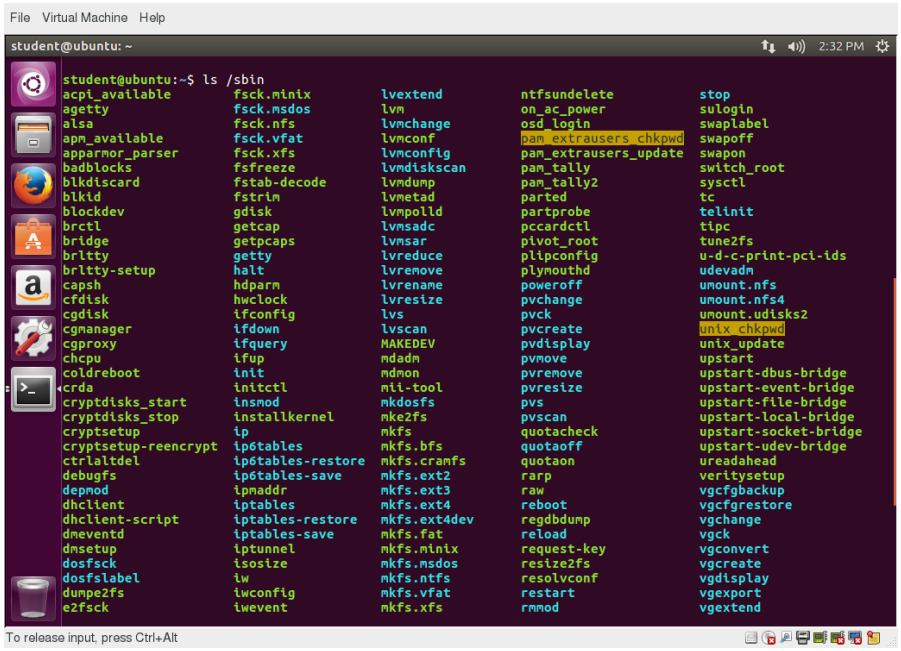


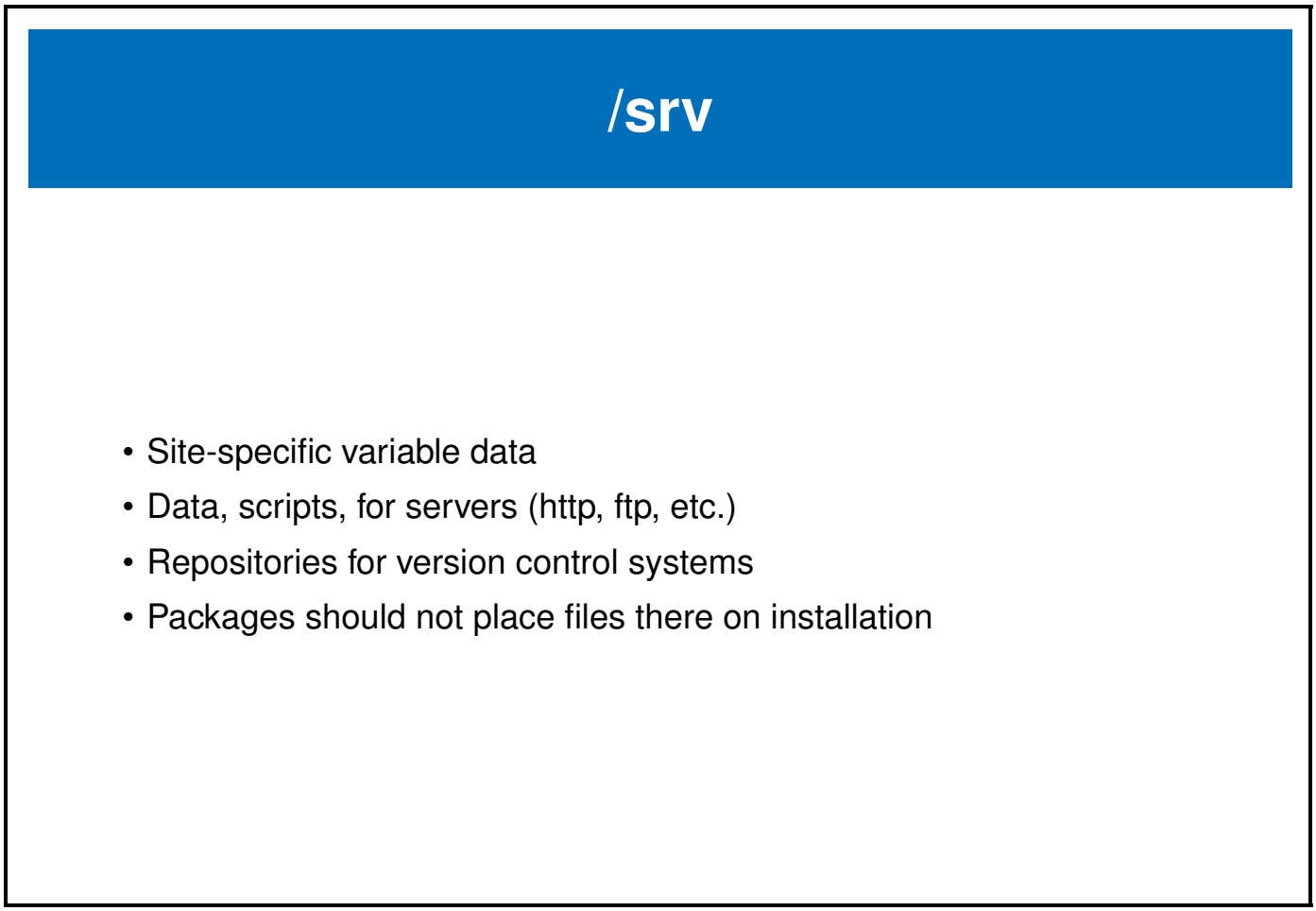
Figure 2.11: **/sbin Directory: Ubuntu 18.04**

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.

© Copyright the Linux Foundation 2023. All rights reserved.

2.18 /srv



According to the **FHS**:

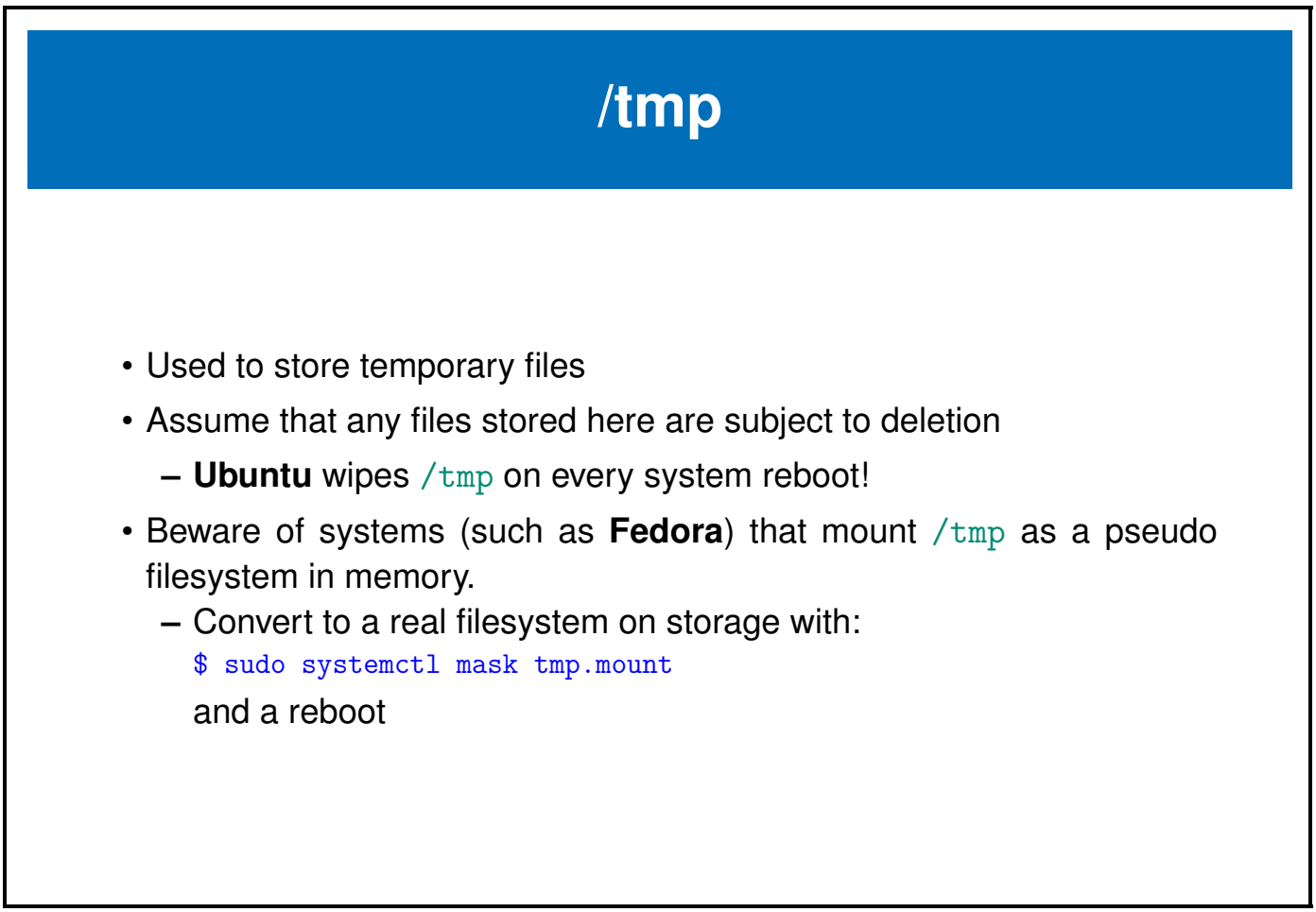
> /srv contains site-specific data which is served by this system.
>
> This main purpose of specifying this is so that users may find the location of the data files for particular service, and so that services which require a single tree for readonly data, writable data and scripts (such as cgi scripts) 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`.

On **Linux** distributions such as **Ubuntu** and **Red Hat**-based ones, `/srv` is empty by default.

© Copyright the Linux Foundation 2023. All rights reserved.

2.19 /tmp



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 **Feodra**, by issuing the command:

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

followed by a system reboot.

© Copyright the Linux Foundation 2023. All rights reserved.

2.20 /usr

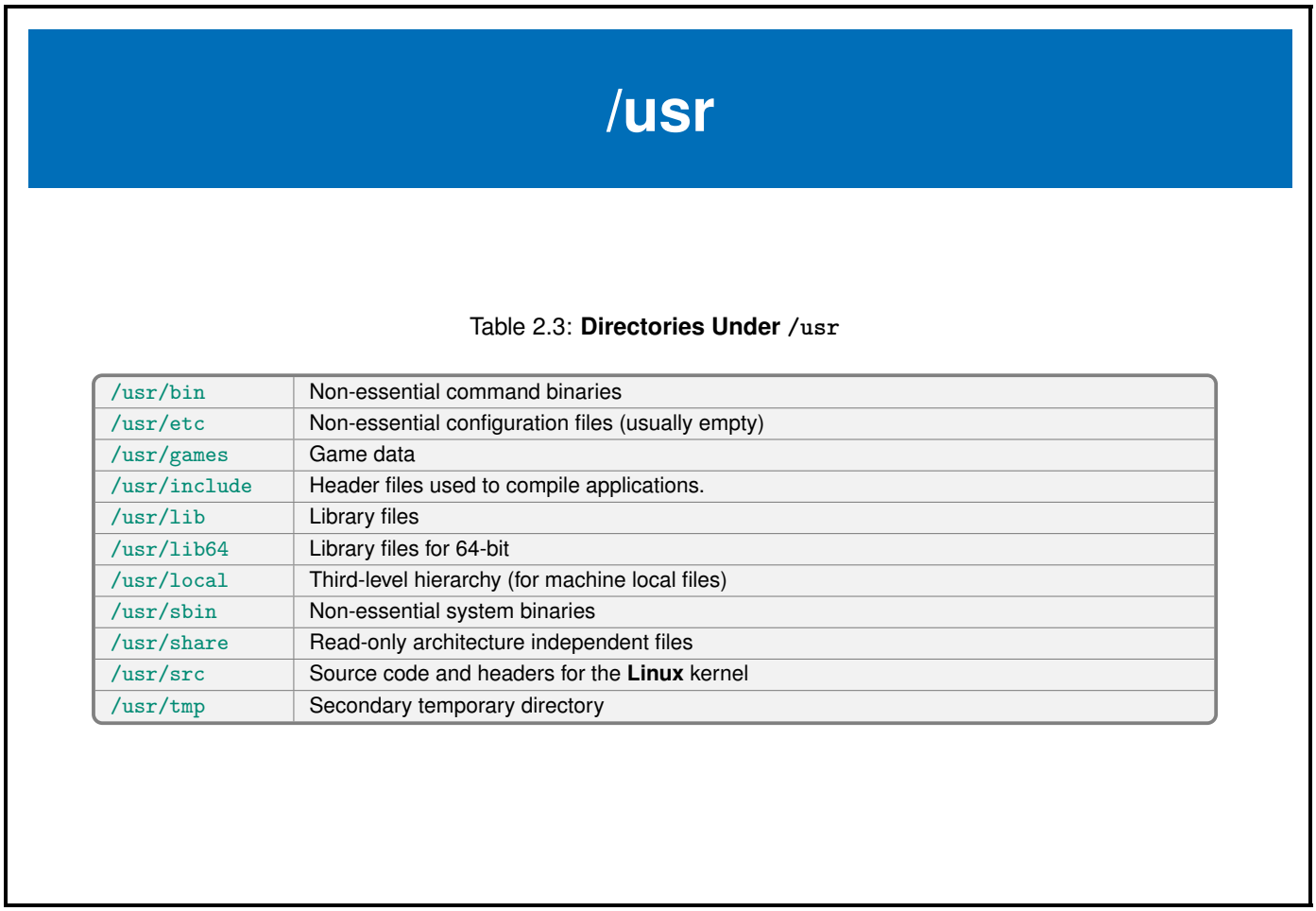


Table 2.3: **Directories Under** `/usr`

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

© Copyright the Linux Foundation 2023. All rights reserved.

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

This directory contains **var**iable (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.

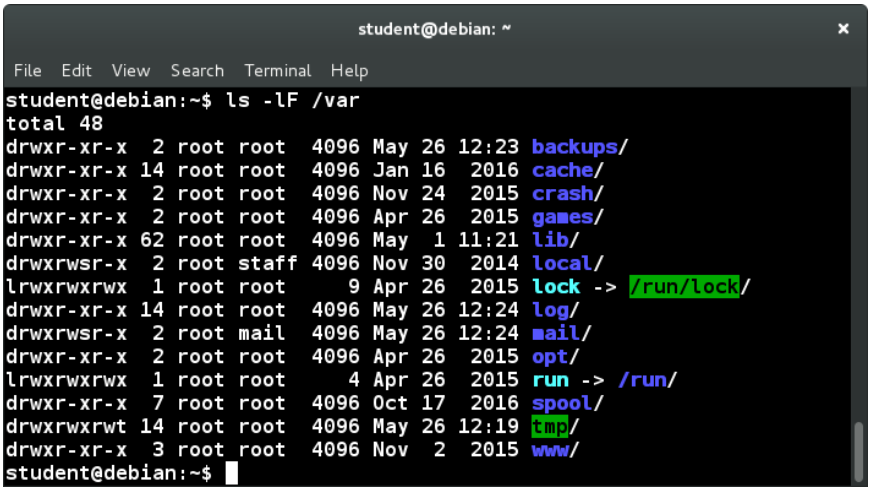


Figure 2.12: /**var Directory**

© Copyright the Linux Foundation 2023. All rights reserved.

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`.

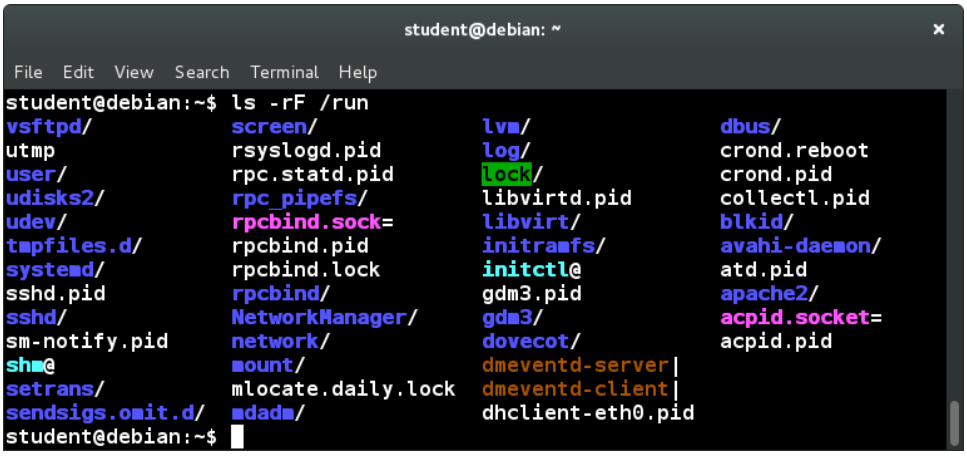


Figure 2.13: **/run Directory**

© Copyright the Linux Foundation 2023. All rights reserved.

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 /
```

```
  4.3M    /home
  16K     /lost+found
  39M     /etc
  4.0K    /srv
  3.6M    /root
  178M    /opt
  138M    /boot
  6.1G    /usr
  1.1G    /var
  16K     /mnt
  4.0K    /media
  869M    /tmp
  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

© Copyright the Linux Foundation 2023. All rights reserved.

Please Note

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

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

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

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

2. View the following files:

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

© Copyright the Linux Foundation 2023. All rights reserved.

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.

3. 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
```

```
attr/        coredump_filter  gid_map     mountinfo   oom_score_adj  sessionid  syscall
autogroup    cpuset           io          mounts      pagemap        setgroups  task/
auxv         cwd@             limits      mountstats  personality    smaps      timerslack_ns
cgroup       environ          loginuid    net/        projid_map     stack      uid_map
clear_refs   exe@             map_files/  ns/         root@          stat       wchan
cmdline      fd/              maps        oom_adj     sched          statm
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`

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 3

User Environment

3.1 Environment Variables 58
3.2 Command History 62
3.3 Command Aliases 66
3.4 Labs 67

Learning Objectives

By the end of this session, you should be able to:

- Understand, use, set, modify, and export **Environment Variables**.
- Examine previous command history and retrieve, edit, and use earlier commands.
- Set **aliases** for frequently used commands and options.

3.1 Environment Variables

Environment Variables

- Associate a name with a variable value

  HOME
  HOST
  PATH

- Can be listed in different formats with:

  ```
  $ env
  $ export
  $ set
  ```

- All variables are prefixed with $ when referenced:

  ```
  $ echo PATH = $PATH
  ```

- Except when they are being defined:

  ```
  $ MYCOLOR=blue
  ```

  Note there can be no spaces around the equal sign!

By listing the environment variables with **env**, **set** or **printenv**, one can see that there are a large number. Many applications and programming languages use them and expect them, failing if they are not present. **Linux** uses them for many things, setting them up when the system starts. You can create and manipulate your own for your own purposes.

```
$ env | head -2
```

```
XDG_VTNR=1
XDG_SESSION_ID=c259
```

```
$ set | head -2
```

```
ABRT_DEBUG_LOG=/dev/null
BASH=/usr/bin/bash
```

```
$ export | head -2
```

```
declare -x BITS="64"
declare -x CCACHE_COMPRESS="1"
```

```
$ echo $PATH
```

```
/usr/local/texlive/2020/bin/x86_64-linux/:/sbin:/usr/lib64/ccache:/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin::/home/bjmoose/bin
```

```
$ echo $HOME
```

```
/home/bjmoose
```


© Copyright the Linux Foundation 2023. All rights reserved.

Some Important Environment Variables

- HOME
  - User's home directory
  - Lets you reach home directory with **cd**
- PATH
  - Ordered list of directories to search for programs to run
  - Directories separated by colons
- PS1
  - Command line prompt, easy to customize
- SHELL
  - User's default shell (**bash**, **csh**, etc.)
- EDITOR
  - User's default editor (**emacs**, **vi**, etc.)

```
$ echo $HOME
```

```
/home/rjsquirrel
```

```
$ echo $PATH
```

```
/usr/lib64/ccache:/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin:/home/rjsquirrel/bin
```

```
$ echo $PS1
```

```
\h:$PWD>
```

```
$ echo $SHELL
```

```
/bin/bash
```

```
$ echo $EDITOR
```

```
/usr/bin/emacs
```


© Copyright the Linux Foundation 2023. All rights reserved.

Setting Environment Variables

- List a specific variable:
  `$ echo $SHELL`
- Set a new variable value:
  `$ VARIABLE=value`
- Add a new variable permanently:
  - Edit `~/.bashrc` to include `VARIABLE=value`
  - Start a new shell or logout/login

```
$ echo $SHELL
```

```
/bin/bash
```

```
$ echo $EDITOR
```

```
/usr/bin/emacs
```

```
$ export EDITOR=/usr/bin/nano
$ echo $EDITOR
```

```
/usr/bin/nano
```

```
$ echo $MYVAR

$ vi .bashrc          <-- added "export MYVAR=myvalue"
$ echo $MYVAR

$ source .bashrc
$ echo $MYVAR
```

```
myvalue
```


© Copyright the Linux Foundation 2023. All rights reserved.

Exporting Environment Variables

- Environment variables are not inherited by child processes by default
- **Exporting** a variable changes that behavior:

```
$ VAR=value ; export VAR
$ export VAR=value
```

- Exported variables can be listed:

```
$ export
```

By default, variables created within a script are only available to the current shell: child processes (sub-shells) will not have access to the content of this variable. In order for variables to be visible to child processes, they need to be **exported** using the **export** command. Exporting a variable can be done in one step:

```
$ export VAR=value
```

or in two steps:

```
$ VAR=value ; export VAR
```

Keep in mind that the child process is allowed to modify exported variables, but the change in this case will not propagate back to the parent shell since exported variables are not shared but only copied.

```
$ export VERSION=$(uname-r)
```

```
$ export
```

```
declare -x BITS="64"
declare -x CCACHE_COMPRESS="1"
declare -x CCACHE_DIR="/mp/.ccache"
...
declare -x VERSION="4.0.3"
...
```


© Copyright the Linux Foundation 2023. All rights reserved.

3.2 Command History

History

- Command to view previous history:
  ```
  $ history
  ```
- Shell stores history in `~/.bash_history`
- Location of the history file: `HISTFILE`
- Maximum number of lines in history file: `HISTFILESIZE`
- Maximum number of lines of history list in current session: `HISTSIZE`

© Copyright the Linux Foundation 2023. All rights reserved.

Display the location of the history file.

```
$ echo $HISTFILE
```

```
/home/jbond/.bash_history
```

Display the maximum size, in lines, of the history file.

```
$ echo $HISTFILESIZE
```

```
2000
```

Display the current size, in lines, of the history file.

```
$ echo $HISTSIZE
```

```
1000
```

Display the first 2 lines in the history file.

```
$ history | head -2
```

```
40  cd RELEASE
41  ls -ltr
```

Recalling, Editing Previous Commands

- Go to previous/recent commands:
  - Up/Down arrow keys
- Execute previous command:
  - !!
- Search previously used commands:
  - CTRL-R
  - Hit multiple times to scan backwards through previous commands.

© Copyright the Linux Foundation 2023. All rights reserved.

With the arrow keys, you are essentially moving up and down your **history** list - most recently executed command first.

`!!` (often pronounced as “bang-bang”) executes the previous command.

Here is an example of `CTRL-R` used to search through history:

```
# This all happens on 1 line
$ CTRL-R  git
(reverse-i-search)`git': git rm -f exported-variables.inc
# Type Enter to execute or hit again:
$ CTRL-R
(reverse-i-search)`git': git ls-files
# Type Enter to execute
```

Previous Commands Using History

- Start a history substitution:
  `!`
- Refer to the last argument in a line:
  `!$`
- Refer to the n-th command line:
  `!n`
- Refer to most recent command starting with `string`
  `!string`

© Copyright the Linux Foundation 2023. All rights reserved.

All history substitutions start with `!`.

Thus, in the command:

```
$ ls -l /bin /etc /var
```

`!$` refers to `/var` which is the last argument in the line above.

```
$ history
```

```
  1  echo $SHELL
  2  echo $HOME
  3  echo $PS1
  4  ls -a
  5  ls -l /etc/passwd
  6  sleep 1000
  7  history
```

```
$ !1              # Execute command #1 above
```

```
  echo $SHELL
  /bin/bash
```

```
$ !sl             # Execute command beginning with "sl"
```

```
  sleep 1000
```


© Copyright the Linux Foundation 2023. All rights reserved.

3.3 Command Aliases

Aliases

- Create customized commands by creating **aliases**

  ```
  $ alias name=command
  ```

  Note there are no spaces around the equal sign
- To make persistent, edit `~/.bashrc`
- Excellent for modifying standard program behavior as in:

  ```
  $ alias rm='rm -i'
  ```
- Get rid of an alias:

  ```
  $ unalias name
  ```
- List currently active aliases:

  ```
  $ alias
  ```

Aliases permit custom definitions. Typing **alias** with no arguments gives the list of defined aliases. **unalias** gets rid of an alias.

Some examples:

```
$ alias l='ls -laF'
$ alias dir='ls -latF'
$ alias rm='rm -i'
$ alias mv='mv -i'
$ alias cp='cp -ipdv'
$ alias df='df -T'
$ alias gitstat='git status -uno'
$ alias diffside='diff --side-by-side --ignore-all-space'
$ alias scapt='gnome-screenshot -i -w'
```


© Copyright the Linux Foundation 2023. All rights reserved.

3.4 Labs

Exercise 3.1: Adding the ~/work directory to your path

Create a small file, ~/work/ls, which contains just the line:

```
echo HELLO, this is the phony ls program.
```

Then make it executable by doing:

```
$ chmod +x ~/work/ls
```

1. Append ~/work to your path, so it is searched only **after** your usual path is considered. Type **ls** and see which program is run: /bin/ls or ~/work/ls.
2. Pre-pend ~/work to your path, so it is searched **before** your usual path is considered. Once again, type **ls** and see which program is run: /bin/ls or ~/work/ls.

What are the security considerations in altering the path this way?

Solution 3.1

First create the phony **ls** program, using an editor, or just simply doing:

```
$ echo "echo HELLO, this is the phony ls program." > ~/work/ls
$ chmod +x ~/work/ls
```

For the next two steps it is a good idea to work in another terminal window, or just start a new shell, so the changes do not persist on later issued commands. You can start a new shell by just typing **bash**.

1. ```
   $ bash
   $ PATH=$PATH:~/work
   ```

   ```
   bin  etc  games  include  lib  lib64  libexec  local  sbin  share  src       tmp
   ```

   ```
   $ exit
   ```

2. ```
   $ bash
   $ PATH=~/work:$PATH
   $ ls /usr
   ```

   ```
   HELLO, this is the phony ls program.
   ```

   ```
   $ exit
   ```

Very Important

Note the second form is a very dangerous thing to do, and is a trivial way to insert a **Trojan Horse** program; if someone can put a malicious program in ~/work, they can trick you into running it accidentally.
Make sure other users don't have **write** permission to directories in your \$PATH.

Exercise 3.2: Command history

© Copyright the Linux Foundation 2023. All rights reserved.

You have been busy working with at your **Linux** workstation long enough to have typed in about 100 commands in one particular **bash** command shell.

At some point earlier, you used a new command, but the exact name has slipped your mind.

Or perhaps it was a pretty complicated command with a bunch of options and arguments and you don't want to go through the error prone process of figuring out what to type again.

How do you ascertain what the command was?

Once you find the command in your history, how do you easily issue the command again without having to type it all in at the prompt?

Solution 3.2

Command history:

The **history** command is the way to display the commands you have typed in:

```
$ history
```

```
    1  cd /
    2  ls
    3  cd
    4  pwd
    5  echo $SHELL
    6  ls /var/
    7  ls /usr/bin
    8  ls /usr/local/bin
    9  man fstab
   10  ls
          . . .
```

In order to re-run a previous command, you have a few choices. Let's say that you wanted to re-run the **man** command you ran way back when you first logged-in. You could type

```
$ !9
```

to re-run the command listed as #9. If this was the only **man** command that you typed in, you could also type

```
$ !man
```

now that you remember the command name that you typed. Finally, if you had typed a few **man** commands, you could use CTRL-R to search backward in your history to find the specific **man** command that you want to re-run, and then just hit Return to execute it.

Exercise 3.3: Command alias

Typing long commands and filenames over and over again gets rather tedious, and leads to a lot of trivial errors such as typos.

Deploying **aliases** allows us to define short cuts to alleviate the pain of all of this typing.

Suppose you are a member of a project team that works in a common, shared directory for your project. This directory is located in /home/staff/RandD/projects/projectX/src.

When you are working on Project X, you often need to create and modify your files in this directory. It doesn't take too long before typing in:

```
$ cd /home/staff/RandD/projects/projectX/src
```

gets tedious.

Define and use an alias named "projx" to do the above **cd** command for you.

© Copyright the Linux Foundation 2023. All rights reserved.

Solution 3.3

Command alias:

The **alias** line would look like this:

```
$ alias projx='cd /home/staff/RandD/projects/projectX/src'
```

Note you can use double quotes instead of single quotes, or use no quotes at all since their is no white space in your defined alias.

All you have to do now to change to the directory is

```
$ projx
```

To make the alias persistent, just place it in your `$HOME/.bashrc` file.

© Copyright the Linux Foundation 2023. All rights reserved.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 4

User Account Management

4.1 User Accounts 72
4.2 Shell Startup Files 75
4.3 Management of User Accounts 78
4.4 Locked Accounts 80
4.5 Passwords 81
4.6 /etc/shadow 82
4.7 Password Management 84
4.8 Password Aging 85
4.9 The root Account 86
4.10 SSH 88
4.11 Labs 95

Learning Objectives

By the end of this session, you should be able to:

- Explain the purpose of individual user accounts and list their main attributes.
- Create new user accounts and modify existing account properties, as well as remove or lock accounts.
- Understand how user passwords are set, encrypted and stored, and how to require changes in passwords over time for security purposes.
- Understand the role of the root account and when to use it.
- Use Secure Shell (**ssh**) and remove logins and commands.

4.1 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.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

Determine the Current User

- Find out users who are currently logged in:
  - **who**
- Find out current user ID:
  - **whoami**
- Find out current user ID, group ID, and their numeric version:
  - **id**

For example:

```
$ who
```

```
user     pts/0        2023-03-09 12:58 (:0)
user     pts/1        2023-03-09 23:23 (:0)
user     pts/2        2023-03-09 23:55 (:0)
user     pts/3        2023-03-09 23:37 (:0)
user     pts/4        2023-03-09 23:38 (:0)
user     pts/5        2023-03-31 09:47 (:0)
user     pts/6        2023-03-31 11:13 (:0)
badguy   pts/8        2023-04-01 16:54 (meow.local)
```

```
$ whoami
```

```
user
```

```
$ id
```

```
uid=1000(user) gid=1000(user) groups=1000(user),999(qubes)
```


© Copyright the Linux Foundation 2023. All rights reserved.

4.2 Shell Startup Files

Startup Files

- Each command shell uses multiple startup files to create environment
- Interactive environments affected somewhat differently
- The files in `/etc` provide global settings
- The files in the user home directory override global settings

Every time a new shell (either a command window or a script that is run) begins executing, there are files included that contain elements employed to ensure proper functioning. This may include:

- Defining relevant **Environment Variables** that are used by many programs and scripts (including `$PATH`).
- Defining **aliases** that are used as shorthand to specify commands and options.
- Defining **functions** that can be used in subsequent scripts.

There are usually system-wide global initializing files found in `/etc` that are usually used by all users before individualized files are used.

Exactly which files are used and in what order depends on the context as we will explain.

© Copyright the Linux Foundation 2023. All rights reserved.

Advantages of Startup Files

- Customize user's prompt
- Set user's terminal type
- Set command line shortcuts and aliases
- Set default text editor

Without the startup (initialization) file processing, each time a command or program is run there may be a lot of set up work to ensure proper functioning.

Many programs evaluate certain environment variables are set when they begin to execute, and then make use of them to control functioning.

For example, any program which needs to modify text files interactively will see how `EDITOR` is set, perhaps to `vim`, `emacs`, or `nano`; whichever editor the user prefers to use.

We will see examples of the above in the section on Environment Variables.

© Copyright the Linux Foundation 2023. All rights reserved.

Startup Files Order

- Default values for all users
  `/etc/profile`
- Login shells configuration
  `~/.bash_profile`
- Login initialization
  `~/.bash_login`
- Overrides `/etc/profile`
  `~/.profile`
- Interactive non-login shells configuration
  `~/.bashrc`
- Usually you only need to customize `~/.bashrc`

When you login to **Linux**, `/etc/profile` is always read and evaluated.

Next, the following files are searched for in this order:

`~/.bash_profile`

`~/.bash_login`

`~/.profile`

After finding the first file it comes to, the **Linux** login shell will evaluate that one startup file and ignore all the rest.

While this may sound redundant, various **Linux** distributions tend to use different startup files.

Every time you create a sub-shell, but aren't logging-in, only `~/.bashrc` is read and evaluated. While it is not read and evaluated with a login shell, most distributions and users will call `~/.bashrc` from within one of the three user-owned startup files; so, in reality, `~/.bashrc` is used for login shells.

Thus the vast majority of your customizations should go into `~/.bashrc`.

© Copyright the Linux Foundation 2023. All rights reserved.

4.3 Management of User Accounts

Creating User Accounts with useradd

- **useradd** allows for default operation

```
$ sudo useradd bjmoose
```

  - Creates account `bjmoose`
  - 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 bjmoose
```

causes the following steps to execute:

- The next available `UID` greater than `UID_MIN` (specified in `/etc/login.defs`) by default is assigned as `bjmoose`'s `UID`.
- A group called `bjmoose` with a `GID=UID` is also created and assigned as `bjmoose`'s primary group.
- A home directory `/home/bjmoose` is created and owned by `bjmoose`.
- `bjmoose`'s login shell will be /**bin**/**bash**.
- The contents of `/etc/skel` is copied to `/home/bjmoose`. 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 `bjmoose`'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.

© Copyright the Linux Foundation 2023. All rights reserved.

Modifying and Deleting User Accounts

- The root user can delete user accounts with **userdel**

  `$ sudo userdel rjsquirrel`

  Deletes the `rjsquirrel` user account, but does not remove her home directory
- User accounts can be modified with **usermod**

  `$ sudo usermod -L bjmoose`

  Locks the account for `bjmoose` so he cannot login

`$ sudo userdel rjsquirrel`

All references to the user `rjsquirrel` 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/rjsquirrel`) 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.

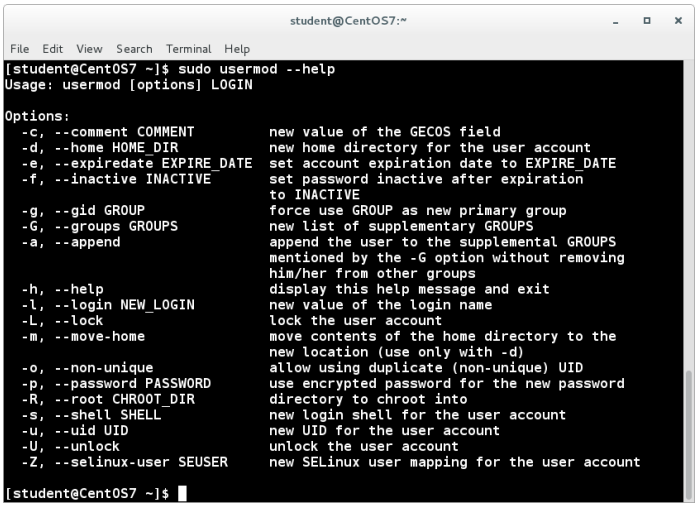


Figure 4.1: **Using usermod**

© Copyright the Linux Foundation 2023. All rights reserved.

4.4 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 "!!" in `/etc/shadow`
- 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 bjmoose
```

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 2001-09-11 rjsquirrel
```

The actual date is irrelevant as long as it is in the past.

© Copyright the Linux Foundation 2023. All rights reserved.

4.5 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 use 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**, or for advanced users - the slightly safer tools **vipw**/**vigr**

© Copyright the Linux Foundation 2023. All rights reserved.

4.6 /etc/shadow

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.

© Copyright the Linux Foundation 2023. All rights reserved.

/etc/shadow format

- /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:$6$iCZyCnBJH9rmq7P.$RYNm10Jg3wrhAtUnahBZ/mTMg.RzQE6iBXyqaXHvxxbK\
   TYqj.d9wpoQFuRp7fPEE3hMK3W2gcIYhiXa9MIA9w1:16316:0:99999:7:::
```

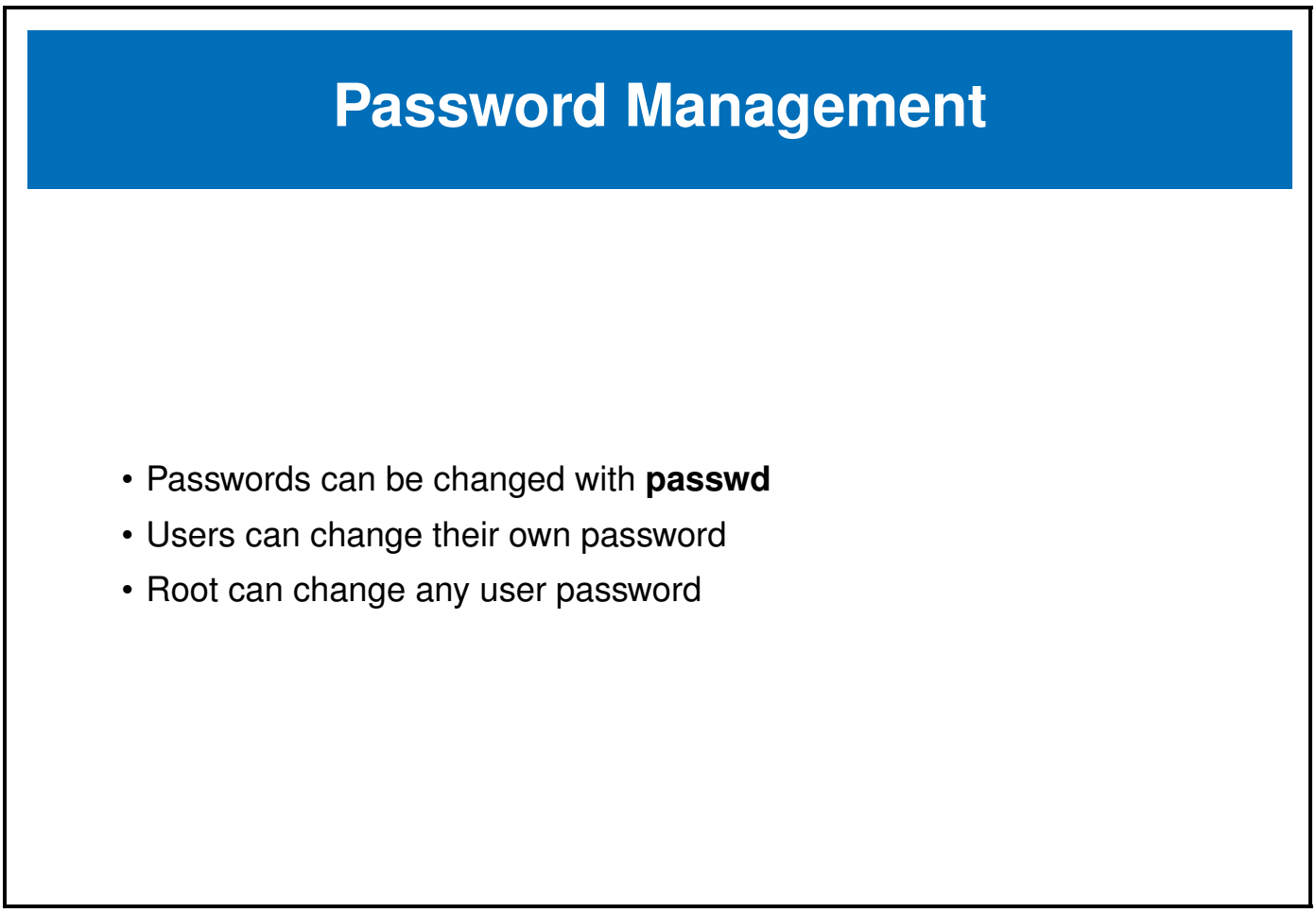
The username in each record must exactly match that found in /etc/passwd, and also must appear in the identical order.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

4.7 Password Management



By default, the password choice is examined by pam_cracklib.so, which furthers making good password choices.
A normal user changing their password:

```
$ passwd
```

```
Changing password for bjmoose
(current) UNIX password: <bjmoose's password>
New UNIX password: <bjmoose's-new-password>
Retype new UNIX password: <bjmoose's-new-password>
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 rjsquirrel
```

```
New UNIX password:  <rjsquirrel's-new-password>
Retype new UNIX password: <rjsquirrel's-password>
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.

© Copyright the Linux Foundation 2023. All rights reserved.

4.8 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 beaver
$ sudo chage -m 14 -M 30 wally
$ sudo chage -E 2012-4-1 eddie
$ sudo chage -d 0 june
```

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
```

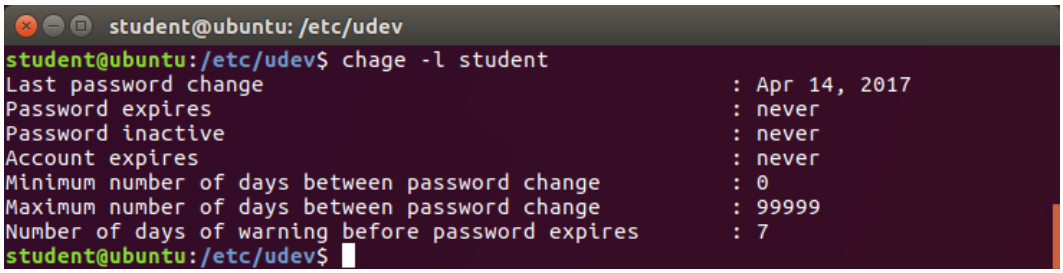


Figure 4.2: **Using chage**

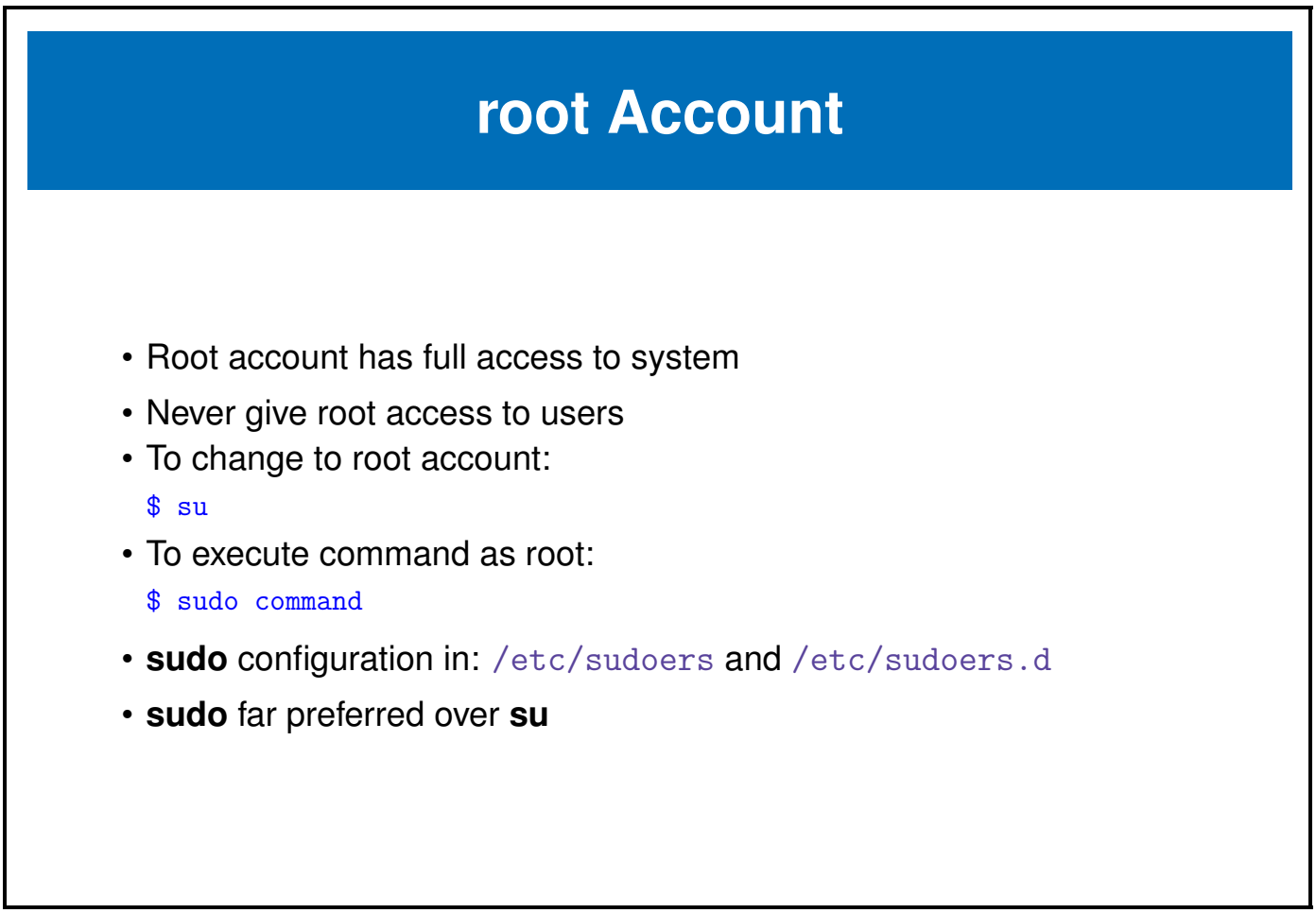
Only the root user can use **chage**. The one exception to this is that any user can run with the `-l` option to determine when their password or account is due to expire.

To force a user to change their password at their next login:

```
$ sudo chage -d 0 USERNAME
```


© Copyright the Linux Foundation 2023. All rights reserved.

4.9 The root Account



root has access to everything and can do everything; this is a very powerful account.

sudo allows regular user accounts to have root privileges on a temporary basis. **sudo** can be configured to allow only certain accounts to have this ability and for certain accounts to only have elevated privileges for certain commands. See **man sudo** for more details.

su (pronounced ess-you and means switch or substitute user) creates a sub-shell environment that allows the user elevated privileges until they exit that shell. **All** commands executed in that sub-shell are executed with the elevated privileges of the root user.

Mistakes will be made! It is best to limit your exposure to big mistakes, like removing many important files, by not using an **su** sub-shell to execute commands that do not require elevated privileges.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

4.10 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: **S**ecure **SH**ell)

```
$ ssh remote_computer.com
$ ssh some_user@remote_computer.com
$ ssh some_user@remote_computer.com apt update
$ scp file.txt remote_computer.com:/tmp
$ scp usr1@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
```

or even easier:

```
$ pssh -viH node1 node2 node3 some_command
```


© Copyright the Linux Foundation 2023. All rights reserved.

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 (**S**ecure **SH**ell) 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
```

```
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.)

You can use the **pssh** (**Parallel SSH**) utility to execute a command on multiple systems in one fell stroke as in:

```
$ pssh -viH machine1 machine2 machine3 do_something
```

You may need to read the **man** page for **pssh** to figure out all its options and how to deal with passwords.

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**

© Copyright the Linux Foundation 2023. All rights reserved.

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
   Generating public/private rsa key pair.
   ...
```

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. Note that `authorized_keys` contains public keys that are allowed to **login as you on this machine**.

```
$ cat authorized_keys
   ssh-rsa AAAAB3Nza........student@debian
   ssh-rsa AAAAB3Nza........student@fedora
```

`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.

```
$ cat known_hosts
   x7 ecdsa-sha2-nistp256 AAAAE2VjZHNhLXN...... M=
   192.168.1.200 ecdsa-sha2-nistp256 AAAA...... bcs=
```


© Copyright the Linux Foundation 2023. All rights reserved.

SSH config file precedence

The order the configuration files are processed:

1. **command line** options
2. `~/.ssh/config`
3. `/etc/ssh/ssh_config`

The precedence of the files and the contents is **first match used**.

/home/student/.ssh/config

```
Host apple
        HostName 192.168.0.196
        User student
        Port 4242
        IdentityFile /home/student/.ssh/custom

Host aws
        Hostname ec2-34-238-135-25.compute-1.amazonaws.com
        User ubuntu
        IdentityFile /home/student/.ssh/cloud1.pem
        ForwardX11 no
        PasswordAuthentication no

Host *
```

In the above `~/.ssh/config` specific configuration information is listed for the hosts **apple** and **aws**. If neither of these match then the generic parameters would apply. This configuration has no generic parameters. The command usage would be:

```
$ ssh apple
```

or

```
$ ssh aws
```


© Copyright the Linux Foundation 2023. All rights reserved.

SSH on a Cloud system

- SSH on a cloud system works the same, with some automation applied.
- The ssh keys are generated for the default user during cloud system creation
- The public key is copied into the default user's `authorized_keys` file
- An option to copy the public and private to your local system is presented
- Password-based authentication is generally not supported for users on cloud systems

It is not uncommon to misplace public keys on remote systems. Recreating a public can be done with **ssh-keygen**:

```
student@ubuntu:~/.ssh$ ~/.ssh$ ls -l
```

```
-rw------- 1 student student 411 Jul 11 07:01 id_ed25519
-rw-r--r-- 1 student student  96 Jul 11 07:01 id_ed25519.pub
```

Rename the public key:

```
student@ubuntu:~/.ssh$ mv id_ed25519.pub id_ed25519.pub.lost
student@ubuntu:~/.ssh$ ls -l
```

```
-rw------- 1 student student 411 Jul 11 07:01 id_ed25519
-rw-r--r-- 1 student student  96 Jul 11 07:01 id_ed25519.pub.lost
```

```
student@ubuntu:~/.ssh$ ssh-keygen -y -f ~/.ssh/id_ed25519 > ~/.ssh/id_ed25519.pub
student@ubuntu:~/.ssh$ ls -l
```

```
-rw------- 1 student student 411 Jul 11 07:01 id_ed25519
-rw-rw-r-- 1 student student  96 Jul 11 07:04 id_ed25519.pub
-rw-r--r-- 1 student student  96 Jul 11 07:01 id_ed25519.pub.lost
```


© Copyright the Linux Foundation 2023. All rights reserved.

Remote Graphical Login

- Login into remote machine with full graphical desktop
- Often use **VNC** (**V**irtual **N**etwork **C**omputing)
- 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
    ```


© Copyright the Linux Foundation 2023. All rights reserved.

To test this first make sure you have the **vnc** packages installed:

```
$ which vncserver vncviewer
```

```
/usr/bin/vncserver
/usr/bin/vncviewer
```

If you do not find these programs you will have to install with something like:

```
$ sudo [dnf|zypper|apt] 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.

© Copyright the Linux Foundation 2023. All rights reserved.

4.11 Labs

Video Demonstration Resources

using_user_accounts_demo.mp4

Exercise 4.1: Accounts

Which account are you logged-in as? How do you find out?

Solution 4.1

Run the **id** utility:

```
$ id
uid=1000(user) gid=1000(user) groups=1000(user),999(qubes)
```

Please Note

There are many ways to accomplish this. Here are some others:
$ whoami
$ grep ^Uid /proc/$$/status

Exercise 4.2: 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 `userp1` on the account to complete this exercise.

Solution 4.2

1. `$ sudo grep student /etc/passwd /etc/shadow`

© Copyright the Linux Foundation 2023. All rights reserved.

```
/etc/passwd:student:x:1000:100:LF Student:/home/student:/bin/bash
/etc/shadow:student:$6$jtoFVPICHhba$iGFFUU8ctrtOGoistJ4/30DrNLi1FS66qnn0VbS6Mvm
  luKI08SgbzT5.IcOHo5j/SOdCagZmF2RgzTvzLb11H0:16028:0:99999:7:::
```

(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`

```
user1@localhost's password:
```

Note you may have to first start up the **sshd** service as in:

`$ sudo service sshd restart`

or

`$ sudo systemctl restart sshd.service`

4. `$ sudo passwd user1`

```
Changing password for user user1.
New password:
```

5. `$ sudo grep user1 /etc/passwd /etc/shadow`

```
/etc/passwd:user1:x:1001:100::/home/user1:/bin/bash
/etc/shadow:user1:$6$OBE1mPMw$CIc7urbQ9ZSnyiniVOeJxKqLFu8fz4whfEexVem2
   TFpucuwRN1CCHZ19XGhj4qVujslRIS.P4aCXd/y1U4utv.:16372:0:99999:7:::
```

6. On either **RHEL** or **openSUSE** systems for example:

`$ cat /etc/default/useradd`

```
# useradd defaults file
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
```

`$ cat /etc/login.defs`

```
....
```

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`

© Copyright the Linux Foundation 2023. All rights reserved.

```
Changing password for user user2.
New password:
```

8. `$ sudo grep user1 /etc/shadow`

```
user1:$6$OBE1mPMw$CIc7urbQ9ZSnyiniVOeJxKqLFu8fz4whfEexVem2TFpucuwRN1CCHZ
   19XGhj4qVujslRIS.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`

```
  user1:$6$OBE1mPMw$CIc7urbQ9ZSnyiniVOeJxKqLFu8fz4whfEexVem2TFpucuwRN1CCHZ
19XGhj4qVujslRIS.P4aCXd/y1U4utv.:16372:0:99999:7::16040:
```

10. `$ sudo usermod -L user1`

    `$ sudo passwd user1`

© Copyright the Linux Foundation 2023. All rights reserved.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 5

Group Management

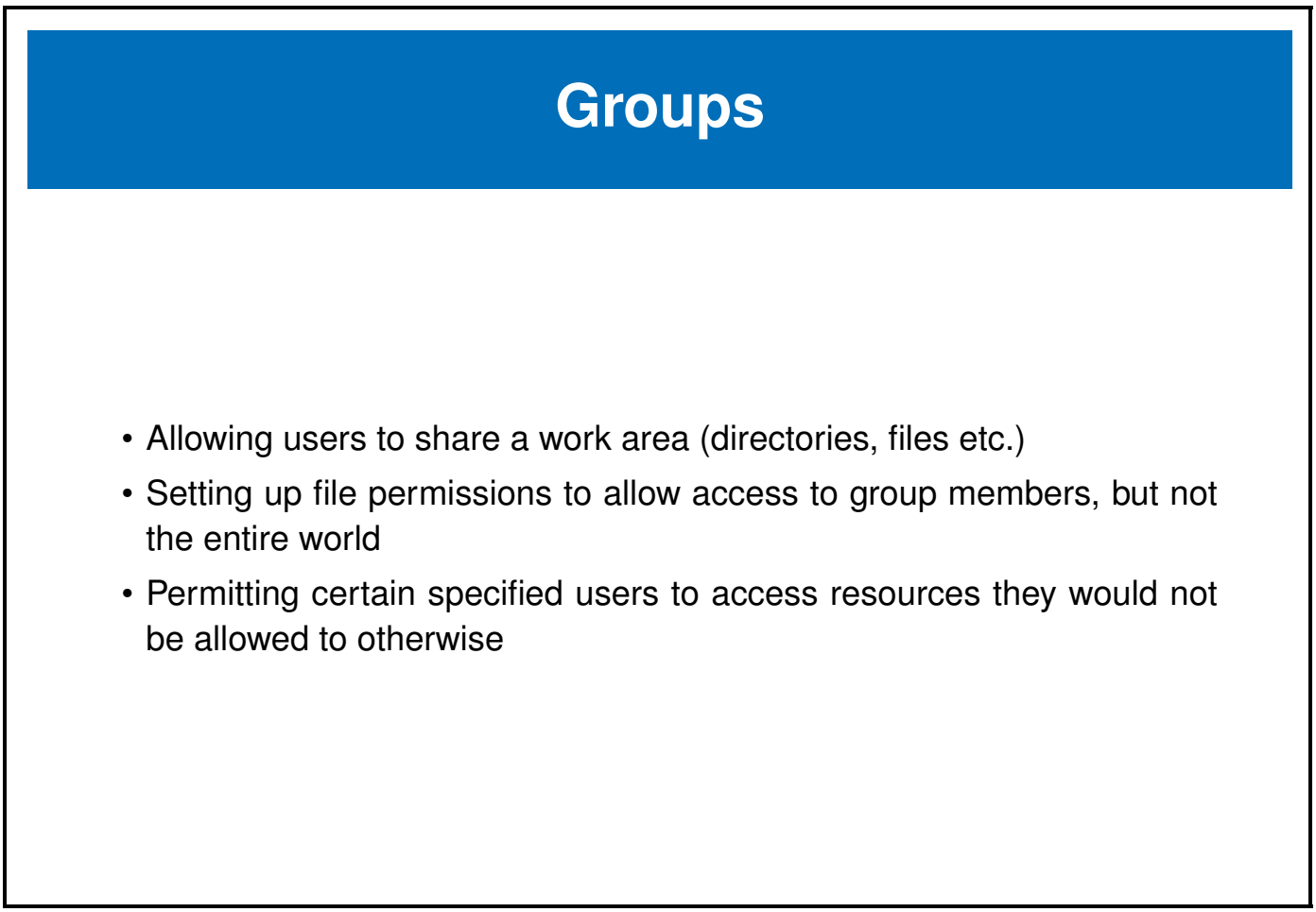
5.1 Groups 100
5.2 Group Membership 101
5.3 Group Management 102
5.4 User Private Groups 103
5.5 Labs 104

Learning Objectives

By the end of this session, you should be able to:

- Explain why it is useful to have Linux users belong to one or more groups.
- Explain the concept of group membership.
- Use **groupadd**, **groupdel**, **groupmod**, and **usermod** to create, remove and manipulate groups and their membership.
- Describe User Private Groups.

5.1 Groups



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.

© Copyright the Linux Foundation 2023. All rights reserved.

5.2 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
```

```
student
```

On **Ubuntu**:

```
student@ubuntu ~$ groups
```

```
student adm cdrom sudo dip plugdev lpadmin sambashare libvirt
```


© Copyright the Linux Foundation 2023. All rights reserved.

5.3 Group Management

Group Management

Group accounts may be managed and maintained with:

- **groupadd**: Add a new group
- **groupmod**: Modify a group's attributes
- **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
```

Very Important

Note: The **usermod -G** command is the total use list of groups, so it will **delete** and add groups all on one command line. Non-destructive use should utilize the **-a** option which will preserve pre-existing group memberships when adding new ones.

© Copyright the Linux Foundation 2023. All rights reserved.

5.4 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.)

© Copyright the Linux Foundation 2023. All rights reserved.

5.5 Labs

Exercise 5.1: 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 5.1

1.
```
$ sudo useradd -m rocky
$ sudo useradd -m bullwinkle
$ sudo passwd rocky
```

```
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

```
$ sudo passwd bullwinkle
```

```
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

```
$ ls -l /home
```

```
total 12
drwxr-xr-x  2 bullwinkle bullwinkle 4096 Oct 30 09:39 bullwinkle
drwxr-xr-x  2 rocky      rocky      4096 Oct 30 09:39 rocky
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
```

```
friends:x:1003:
bosses:x:490:
```


© Copyright the Linux Foundation 2023. All rights reserved.

3.
```
$ sudo usermod -G friends,bosses rocky
$ sudo usermod -G friends bullwinkle
```

```
$ grep -e rocky -e bullwinkle /etc/group
```

```
rocky:x:1001:
bullwinkle:x:1002:
friends:x:1003:rocky,bullwinkle
bosses:x:490:rocky
```

```
$ groups rocky bullwinkle
```

```
rocky : rocky friends bosses
bullwinkle : bullwinkle friends
```

4.
```
$ ssh rocky@localhost
$ cd ~
$ mkdir somedir
$ chgrp bosses somedir
$ ls -l
```

```
total 16
-rw-r--r-- 1 rocky rocky  8980 Oct  4  2013 examples.desktop
drwxrwxr-x 2 rocky bosses 4096 Oct 30 09:53 somedir
```

```
$ chmod a+x .
```

5.
```
$ ssh bullwinkle@localhost
$ touch /home/rocky/somedir/somefile
```

```
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)

© Copyright the Linux Foundation 2023. All rights reserved.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 6

File Permissions and Ownership

6.1 File Permissions and Ownership 108
6.2 File Access Rights 109
6.3 chmod, chown and chgrp 110
6.4 umask 113
6.5 Filesystem ACLs 115
6.6 Labs 116

Learning Objectives

By the end of this session, you should be able to:

- Explain the concepts of owner, group, and world.
- Set file access rights (read, write, and execute) for each category.
- Authenticate requests for file access, respecting proper permissions.
- Use **chmod** to change file permissions, **chown** to change user ownership, and **chgrp** to change group ownership.
- Understand the role of **umask** in establishing desired permissions on newly created files.
- Use **ACL**s to extend the simpler user, group, world and read, write, execute model.

6.1 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
```

```
-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`.

© Copyright the Linux Foundation 2023. All rights reserved.
THE LINUX FOUNDATION Training & Certification

6.2 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.

© Copyright the Linux Foundation 2023. All rights reserved.

6.3 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+rx`.

© Copyright the Linux Foundation 2023. All rights reserved.

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:

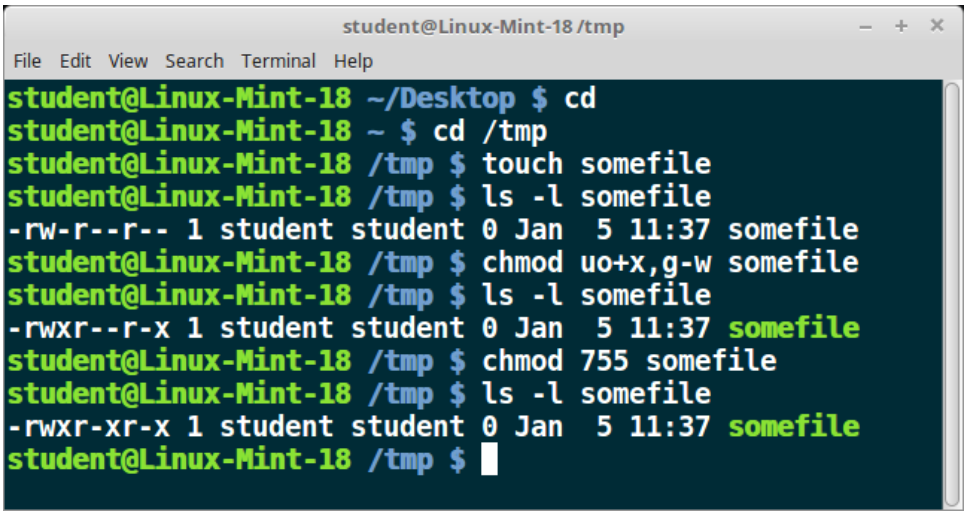


Figure 6.1: **Using chmod**

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

6.4 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

© Copyright the Linux Foundation 2023. All rights reserved.

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
```

```
drwxrwxr-x 2  coop coop 4096 Sep 16 11:18 adir
-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
```

```
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
```


© Copyright the Linux Foundation 2023. All rights reserved.

6.5 Filesystem ACLs

Filesystem ACLs

- **POSIX ACL**s (**A**ccess **C**ontrol **L**ists) 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 **ACL**s
- 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 **ACL**s, 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 **ACL**s 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 **ACL**s.

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
```


© Copyright the Linux Foundation 2023. All rights reserved.

6.6 Labs

Video Demonstration Resources

using_umask_demo.mp4

Exercise 6.1: 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,o+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 6.2: umask

Create an empty file with:

```
$ touch afile
$ ls -l afile
```

```
-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
```

```
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*
```


© Copyright the Linux Foundation 2023. All rights reserved.

Exercise 6.3: Using Access Control Lists

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. Use **setfacl** to make the file writeable by the new user and try again.
5. Use **setfacl** to make the file not readable by the new user and try again.
6. Clean up as necessary.

Solution 6.3

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
```

```
getfacl: Removing leading '/' from absolute path names
# file: tmp/afile
# owner: coop
# group: coop
user::rw-
group::rw-
other::r--
```

2. In window 1:

```
$ sudo useradd fool
$ sudo passwd fool
...
```

3. In window 2:

```
$ sudo su - fool
$ echo another line > /tmp/afile
```

```
-bash: /tmp/afile: Permission denied
```

4. In window 1:

```
$ setfacl -m u:fool:rw /tmp/afile
$ getfacl /tmp/afile
```

```
getfacl: Removing leading '/' from absolute path names
# file: tmp/afile
# owner: coop
# group: coop
user::rw-
user:fool:rw-
group::rw-
mask::rwx
other::r--
```


© Copyright the Linux Foundation 2023. All rights reserved.

In window 2:

```
$ echo another line > /tmp/afile
```

5. In window 1:

```
$ 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
```


© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 7

Package Management Systems

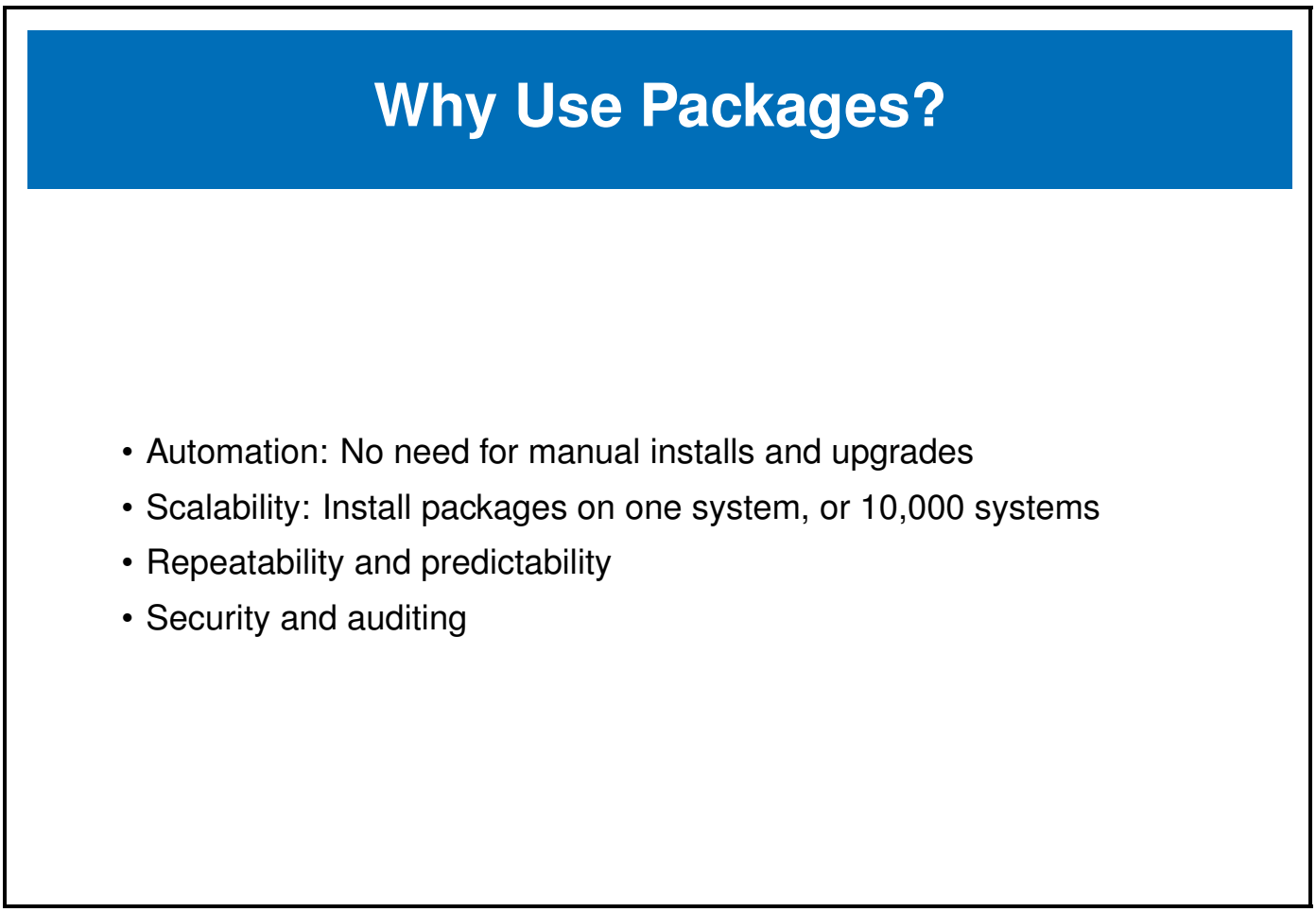
7.1 Why Use Packages? . . . 120
7.2 Software Packaging Concepts . . . 121
7.3 Package Types . . . 122
7.4 Available Package Management Systems . . . 123
7.5 Packaging Tool Levels and Varieties . . . 124
7.6 Package Sources . . . 125
7.7 Creating Software Packages . . . 126
7.8 Revision Control Systems . . . 127
7.9 Available Source Control Systems . . . 128
7.10 The Linux Kernel and git . . . 129
7.11 Labs . . . 131

Learning Objectives

By the end of this session, you should be able to:

- Explain why software package management systems are necessary.
- Understand the function of both binary and source packages.
- Enumerate the main available package management systems.
- Understand why two levels of utilities are needed: one that deals with just bare packages, and one that deals with dependencies among packages.
- Explain how creating your own package enhances the control you have over exactly what goes in software and how it is installed.
- Understand the role of source control systems, and git in particular.

7.1 Why Use Packages?



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.

© Copyright the Linux Foundation 2023. All rights reserved.

7.2 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.

© Copyright the Linux Foundation 2023. All rights reserved.

7.3 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"
```

```
./RPMS/x86_64/p7zip-plugins-16.02-16.el8.x86_64.rpm
./RPMS/x86_64/p7zip-debugsource-16.02-16.el8.x86_64.rpm
./RPMS/x86_64/p7zip-plugins-debuginfo-16.02-16.el8.x86_64.rpm
./RPMS/x86_64/p7zip-16.02-16.el8.x86_64.rpm
./RPMS/noarch/p7zip-doc-16.02-16.el8.noarch.rpm
```

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

© Copyright the Linux Foundation 2023. All rights reserved.

7.4 Available Package Management Systems

Available Package Management Systems

Figure 7.1: **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.

© Copyright the Linux Foundation 2023. All rights reserved.

7.5 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.

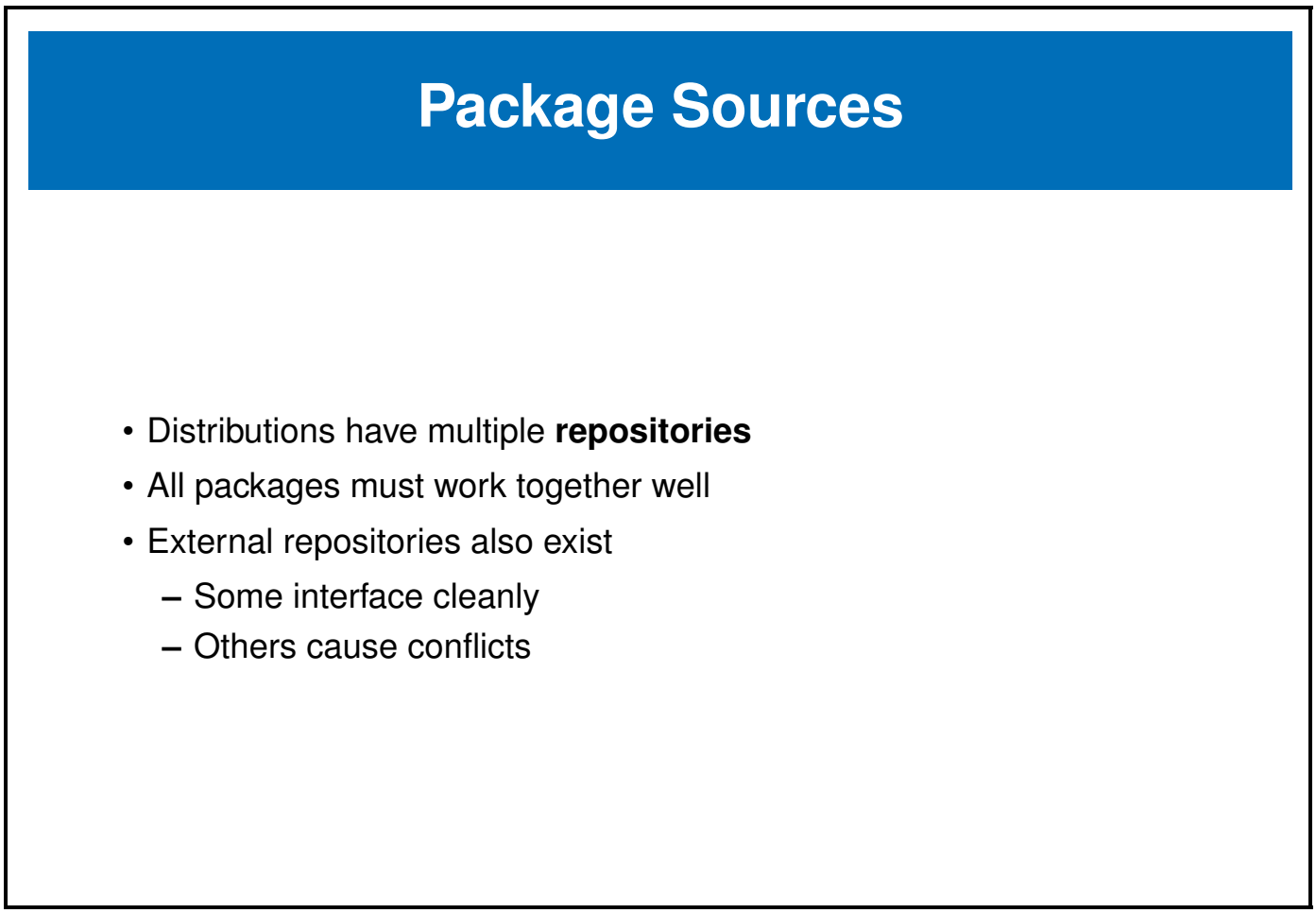
   The **rpm** and **dpkg** utilities play this role for the packaging systems that use them.
2. **High Level Utility**: This solves the dependency problems:
   - If another package or group of packages needs to be installed before software can be installed, such needs will be satisfied.
   - If removing 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** 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.

© Copyright the Linux Foundation 2023. All rights reserved.

7.6 Package Sources



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***.

© Copyright the Linux Foundation 2023. All rights reserved.

7.7 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 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.

© Copyright the Linux Foundation 2023. All rights reserved.

7.8 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 include 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.

© Copyright the Linux Foundation 2023. All rights reserved.

7.9 Available Source Control Systems

Available Source Control Systems

Table 7.1: **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.

© Copyright the Linux Foundation 2023. All rights reserved.

7.10 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

We will have more detailed discussion of **git** and its use in later sections. Here, and in the accompanying lab exercise, we will get a feel for how it works and how to use it.

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/.

© Copyright the Linux Foundation 2023. All rights reserved.

How git Works

- Not a traditional version controls 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**.

© Copyright the Linux Foundation 2023. All rights reserved.

7.11 Labs

Video Demonstration Resources

using_package_management_demo.mp4

Exercise 7.1: Version Control with git

We will have more detailed discussion of **git** in later sections. Here, we will just get a basic feel for how to use it.

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 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 easy 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
   ```

   ```
   total 40
   drwxrwxr-x 2 coop coop 4096 Dec 30 13:59 branches/
   -rw-rw-r-- 1 coop coop   92 Dec 30 13:59 config
   -rw-rw-r-- 1 coop coop   58 Dec 30 13:59 description
   -rw-rw-r-- 1 coop coop   23 Dec 30 13:59 HEAD
   drwxrwxr-x 2 coop coop 4096 Dec 30 13:59 hooks/
   drwxrwxr-x 2 coop coop 4096 Dec 30 13:59 info/
   drwxrwxr-x 4 coop coop 4096 Dec 30 13:59 objects/
   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
   ```


© Copyright the Linux Foundation 2023. All rights reserved.

```
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

      new file: somejunkfile
```

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

5. 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.

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

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

```
diff --git a/somejunkfile b/somejunkfile
index 9638122..6023331 100644
--- a/somejunkfile
+++ b/somejunkfile
@@ -1 +1,2 @@
 some junk
+another line
```

7. To actually commit the changes to the repository we do:

```
$  git commit -m "My initial commit"
```

```
Created initial commit eafad66: My initial commit
 1 files changed, 1 insertions(+), 0 deletions(-)
 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`.

8. You can see your history with:

```
$ git log
```

```
commit eafad66304ebbcd6acfe69843d246de3d8f6b9cc
Author: A Genius <a_genius@linux.com>
Date:   Wed Dec 30 11:07:19 2009 -0600

    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.

© Copyright the Linux Foundation 2023. All rights reserved.

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

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

© Copyright the Linux Foundation 2023. All rights reserved.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 8

dpkg

8.1 DPKG (Debian Package) 136
8.2 Package File Names and Source 137
8.3 DPKG Queries 138
8.4 Installing/Upgrading/Uninstalling 139
8.5 Labs 140

Learning Objectives

By the end of this session, you should be able to:

- Discuss the methods and use of Debian Packaging System (**DPKG**).
- Explain the naming conventions used for both binary and source `.deb` files.
- Know what source packages look like.
- Use querying and verifying operations on packages.
- Install, upgrade, and uninstall **Debian** packages.

8.1 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.

© Copyright the Linux Foundation 2023. All rights reserved.

8.2 Package File Names and Source

Package File Names and Source

A

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:

Figure 8.1: **Getting Source Packages on Ubuntu**

© Copyright the Linux Foundation 2023. All rights reserved.

8.3 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
```

It is often important to get information about a particular package or to locate any files that have changed after installation. For example, to see what version of a particular package is installed:

```
$ dpkg -s dpkg | grep -i version
```

```
Version: 1.19.7ubuntu3.2
```

on **Ubuntu 20.04**.

Without additional arguments, `dpkg -V` will verify all packages on the system:

```
$ sudo dpkg -V
```

```
??5?????? c /etc/logrotate.conf
??5?????? c /etc/logrotate.d/apt
??5?????? c /etc/logrotate.d/bootlog
??5?????? c /etc/logrotate.d/rsyslog
??5?????? c /etc/logrotate.d/ufw
??5?????? c /etc/logrotate.d/apport
??5?????? c /etc/logrotate.d/speech-dispatcher
??5?????? c /etc/logrotate.d/unattended-upgrades
....
```

See the **man** page to interpret the output.

© Copyright the Linux Foundation 2023. All rights reserved.

8.4 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**.

© Copyright the Linux Foundation 2023. All rights reserved.

8.5 Labs

Video Demonstration Resources

using_dpkg_demo.mp4

Exercise 8.0: 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 8.1: 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 8.1

1. `$ dpkg -S logrotate.conf`

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

2. `$ dpkg -L logrotate`

```
...
```

3. `$ dpkg -V logrotate`

4. `$ sudo dpkg -r logrotate`

```
dpkg: dependency problems prevent removal of logrotate:
 ubuntu-standard depends on logrotate; however:
  Package logrotate is to be removed.
 libvirt-daemon-system depends on logrotate.

dpkg: error processing package logrotate (--remove):
 dependency problems - not removing
Errors were encountered while processing:
 logrotate
```


© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 9

APT

9.1 APT 142
9.2 APT Utilities 143
9.3 Queries 144
9.4 Installing/Removing/Upgrading Packages 145
9.5 Cleaning Up 146
9.6 Labs 147

Learning Objectives

By the end of this session, you should be able to:

- Explain what **APT** is.
- Delineate the major **APT** utility programs.
- Perform package queries.
- Install, remove, and upgrade packages using apt
- Clean up and compact system information about packages.

9.1 APT

What is APT?

- Stands for **A**dvanced **P**ackaging **T**ool
- Includes utilities such as **apt** 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 sometimes 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** (**A**dvanced **P**ackaging **T**ool) 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**.

© Copyright the Linux Foundation 2023. All rights reserved.

9.2 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.

© Copyright the Linux Foundation 2023. All rights reserved.

9.3 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 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
```

Display 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.

© Copyright the Linux Foundation 2023. All rights reserved.

9.4 Installing/Removing/Upgrading Packages

Installing/Removing/Upgrading Packages

- Install and remove packages:

```
$ sudo apt install glibc
$ sudo apt remove glibc
$ sudo apt --purge remove glibc
```

- **update** repository databases and then **upgrade** system or packages:

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt dist-upgrade
```

No individual pakage uprades

Unlike with **dnf** or **zypper**, one never updates/upgrades individual packages; the entire package list is dealt with as a unit.

```
$ sudo apt install [package]
```

Used to install new packages or update a package which is already installed.

```
$ sudo apt remove [package]
```

Used to remove a package from the system. This does not remove the configuration files.

```
$ sudo apt --purge remove [package]
```

Used to remove a package and its configuration files from a system.

```
$ sudo apt 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 upgrade
$ sudo apt 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.

© Copyright the Linux Foundation 2023. All rights reserved.

9.5 Cleaning Up

Cleaning Up

- Get rid of any packages not needed anymore, such as older **Linux** kernel versions:

```
$ sudo apt autoremove
```

- Clean out cache files and any archived package files that have been installed:

```
$ sudo apt clean
```

  This can save a lot of space.

Using **apt clean** and **apt autoremove** is a quick and fairly safe way to free up disk space in `/var` when disk space is low.

© Copyright the Linux Foundation 2023. All rights reserved.

9.6 Labs

Video Demonstration Resources

using_package_gui_ubuntu_demo.mp4

Exercise 9.0: 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 9.1: 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 9.1

1. First synchronize the package index files with remote repositories:

   ```
   $ sudo apt update
   ```

   To actually upgrade:

   ```
   $ sudo apt upgrade
   $ sudo apt -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 install apache2-dev
   ```

Exercise 9.2: Using APT to Find Information About a Package

Using **apt-cache** and **apt** (and not **dpkg**), find:

1. All packages that contain a reference to **bash** in their name or description.
2. Installed and available **bash** packages.

© Copyright the Linux Foundation 2023. All rights reserved.

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 9.2

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 9.3: 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 of available **metapackages**:

```
$ apt-cache search metapackage
```

```
bacula - network backup service - metapackage
bacula-client - network backup service - client metapackage
bacula-server - network backup service - server metapackage
cloud-utils - metapackage for installation of upstream cloud-utils source
compiz - OpenGL window and compositing manager
emacs - GNU Emacs editor (metapackage)
....
```

You can then easily install them like regular single packages, as in:

```
$ sudo apt install bacula-client
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  bacula-common bacula-console bacula-fd bacula-traymonitor
Suggested packages:
  bacula-doc kde gnome-desktop-environment
The following NEW packages will be installed:
  bacula-client bacula-common bacula-console bacula-fd bacula-traymonitor
0 upgraded, 5 newly installed, 0 to remove and 0 not upgraded.
Need to get 742 kB of archives.
After this operation, 1,965 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Select an uninstalled metapackage and then remove it.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 10

RPM

10.1 RPM (Red Hat Package Manager . . . 150
10.2 Package File Names . . . 151
10.3 RPM Database and Helper Programs . . . 152
10.4 Queries . . . 154
10.5 Verifying Packages . . . 155
10.6 Installing and Removing Packages . . . 157
10.7 Updating, Upgrading and Freshening RPM Packages . . . 159
10.8 Upgrading the Linux Kernel . . . 161
10.9 rpm2archive and rpm2cpio . . . 162
10.10 Labs . . . 163

Learning Objectives

By the end of this session, you should be able to:

- Understand how the RPM system is organized and what major operations the **rpm** program can accomplish.
- Explain the naming conventions used for both binary and source **rpm** files.
- Query, verify, install, uninstall, upgrade and freshen packages.
- Grasp why new kernels should be installed, rather than upgraded.
- Use **rpm2cpio** to copy packaged files into a **cpio** archive, as well as to extract the files without installing them.

10.1 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 **RPM**s for different architectures

RPM is a package management utility developed by **Red Hat**. (The name originally stood for **R**edhat **P**ackage **M**anager.) 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.

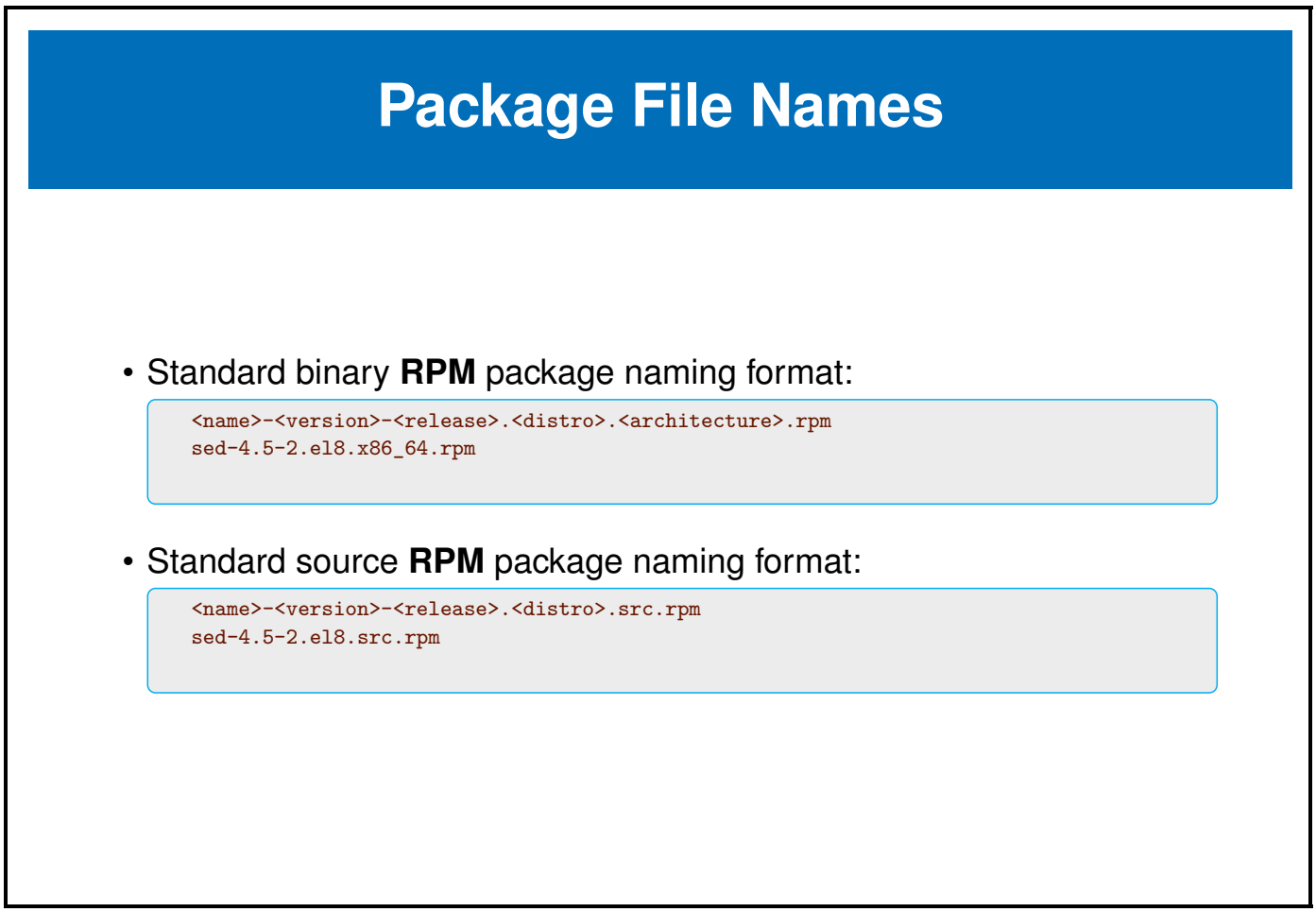
Installation from the Network

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**.

© Copyright the Linux Foundation 2023. All rights reserved.

10.2 Package File Names



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**.

© Copyright the Linux Foundation 2023. All rights reserved.

10.3 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
```


© Copyright the Linux Foundation 2023. All rights reserved.

/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
  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.

© Copyright the Linux Foundation 2023. All rights reserved.

10.4 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 10.1: **rpm Query Command Examples**

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

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
```


© Copyright the Linux Foundation 2023. All rights reserved.

10.5 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
```


© Copyright the Linux Foundation 2023. All rights reserved.

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
```

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

Output indicating that a file is missing.

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

```
S.5....T.  c /etc/logrotate.conf
missing     /usr/sbin/logrotate
```


© Copyright the Linux Foundation 2023. All rights reserved.

10.6 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.

© Copyright the Linux Foundation 2023. All rights reserved.

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
```

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

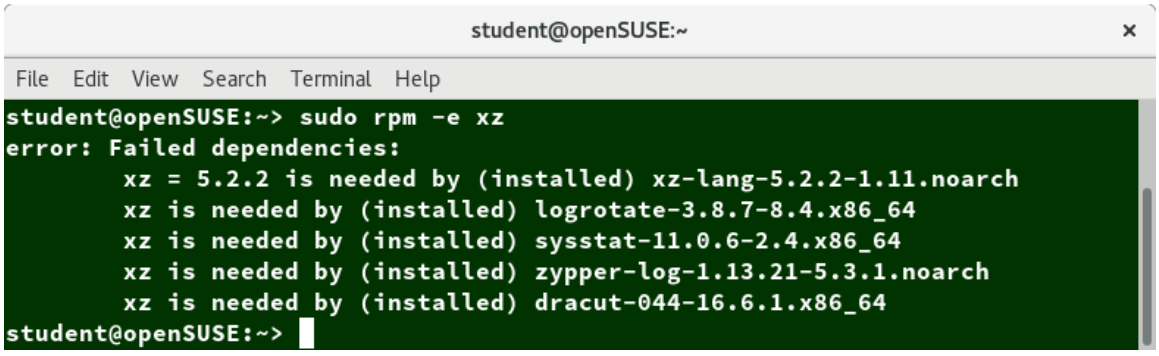


Figure 10.1: **Uninstalling RPM Packages**

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.

© Copyright the Linux Foundation 2023. All rights reserved.

10.7 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.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

10.8 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

Removing Older Package Versions

If you really want to remove older versions of kernel and other packages, on **Red Hat**-based systems you can do:

```
$ sudo dnf remove --oldinstallonly
```

Be careful when using this command.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

10.9 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
```


© Copyright the Linux Foundation 2023. All rights reserved.

10.10 Labs

Video Demonstration Resources

using_rpm_demo.mp4

Exercise 10.0: RPM Required

On Red Hat, 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 10.1: 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 10.1

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

   ```
   logrotate-3.14.0-3.el8.x86_64
   ```

2. `$ rpm -qil logrotate`

   ```
   ...
   ```

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

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

3. `$ rpm -V logrotate`

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

On Red Hat

4. On **RHEL 8**:
   `$ sudo rpm -e logrotate`

© Copyright the Linux Foundation 2023. All rights reserved.

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

On openSUSE

On **openSUSE-Leap 15.1**:

```
$ sudo  rpm -e logrotate
```

```
error: Failed dependencies:
        logrotate is needed by (installed) xdm-1.1.11-lp151.13.2.x86_64
        logrotate is needed by (installed) wpa_supplicant-2.6-lp151.4.4.x86_64
        logrotate is needed by (installed) chrony-3.2-lp151.8.6.x86_64
        logrotate is needed by (installed) net-snmp-5.7.3-lp151.7.5.x86_64
        logrotate is needed by (installed) syslog-service-2.0-lp151.3.3.noarch
        logrotate is needed by (installed) vsftpd-3.0.3-lp151.6.3.x86_64
        logrotate is needed by (installed) libvirt-daemon-5.1.0-lp151.7.6.1.x86_64
        logrotate is needed by (installed) iscsiuio-0.7.8.2-lp151.13.6.1.x86_64
        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 10.2: 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 10.2

1.
```
$ cd /var/lib
$ sudo cp -a rpm rpm_BACKUP
```

2.
```
$ sudo rpm --rebuilddb
```

3.
```
$ ls -l rpm rpm_BACKUP
```


© Copyright the Linux Foundation 2023. All rights reserved.

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`

© Copyright the Linux Foundation 2023. All rights reserved.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 11

dnf and yum

11.1 dnf 168
11.2 yum 169
11.3 Queries 170
11.4 Installing/Removing/Upgrading Packages 171
11.5 Additional dnf Commands 172
11.6 Labs 173

Learning Objectives

By the end of this session, you should be able to:

- Discuss package installers and their characteristics.
- Explain how **dnf** works as a high-level package management system.
- Configure **dnf** to use repositories.
- Discuss the queries **dnf** can be used for.
- Verify, install, remove, and upgrade packages using **dnf**.
- Learn about additional commands and how to install new repositories.

11.1 dnf

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**, **Fedora** and others

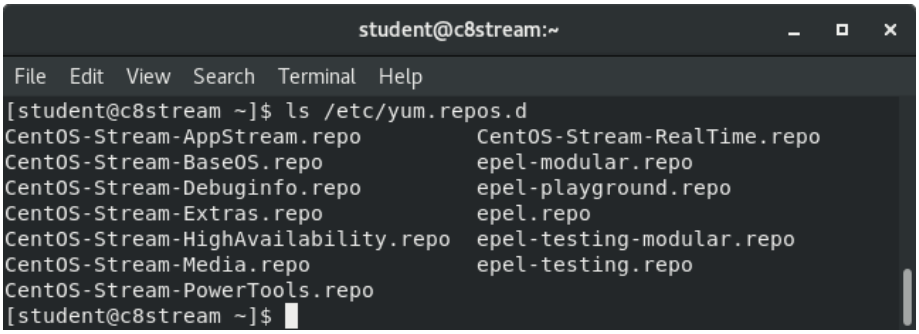


Figure 11.1: **rpm Repositories**

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 | rpmdb | 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.

© Copyright the Linux Foundation 2023. All rights reserved.

11.2 yum

yum

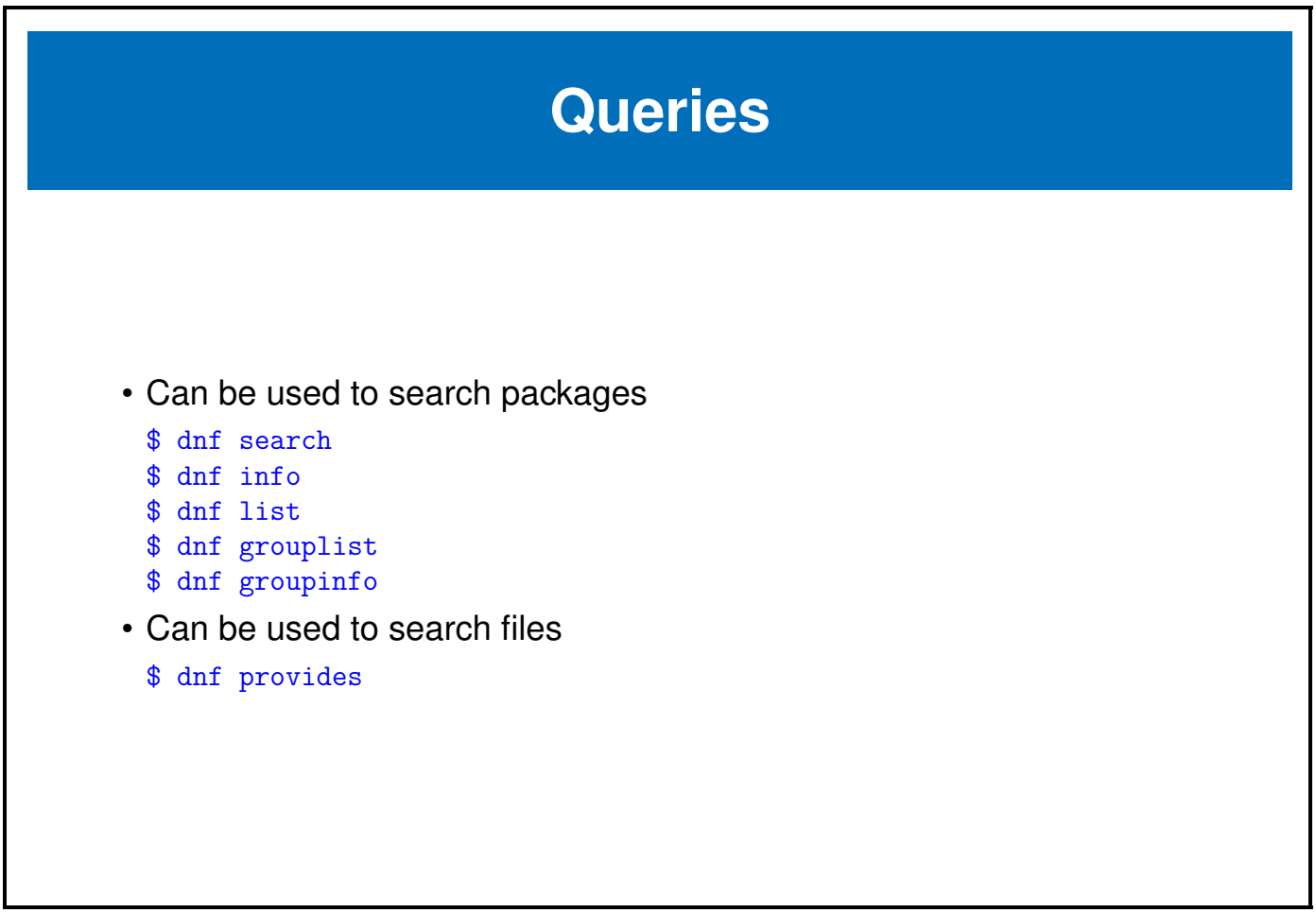
- **dnf** replaced **yum** during the **RHEL/CentOS 7** to **8** transition.
- **Fedora** has used it even longer
- **dnf** is backwards compatible:
  - Almost all common **yum** commands still work
- For more documentation, see:
  https://docs.fedoraproject.org/en-US/quick-docs/dnf/
  https://dnf.readthedocs.io/en/latest/

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.

© Copyright the Linux Foundation 2023. All rights reserved.

11.3 Queries




```
$ 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.)

© Copyright the Linux Foundation 2023. All rights reserved.

11.4 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.

© Copyright the Linux Foundation 2023. All rights reserved.

11.5 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.

© Copyright the Linux Foundation 2023. All rights reserved.

11.6 Labs

Exercise 11.0: dnf or yum Required

On Red Hat, 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 11.1: 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 11.1

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 11.2: 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 11.2

© Copyright the Linux Foundation 2023. All rights reserved.

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 11.3: 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 11.4: 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 to 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
   ```


© Copyright the Linux Foundation 2023. All rights reserved.

```
mirrorlist=http://download.webmin.com/download/yum/mirrorlist
enabled=1
gpgcheck=0
```

2. Install the webmin package.

```
$ sudo dnf install webmin
```


© Copyright the Linux Foundation 2023. All rights reserved.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 12

zypper

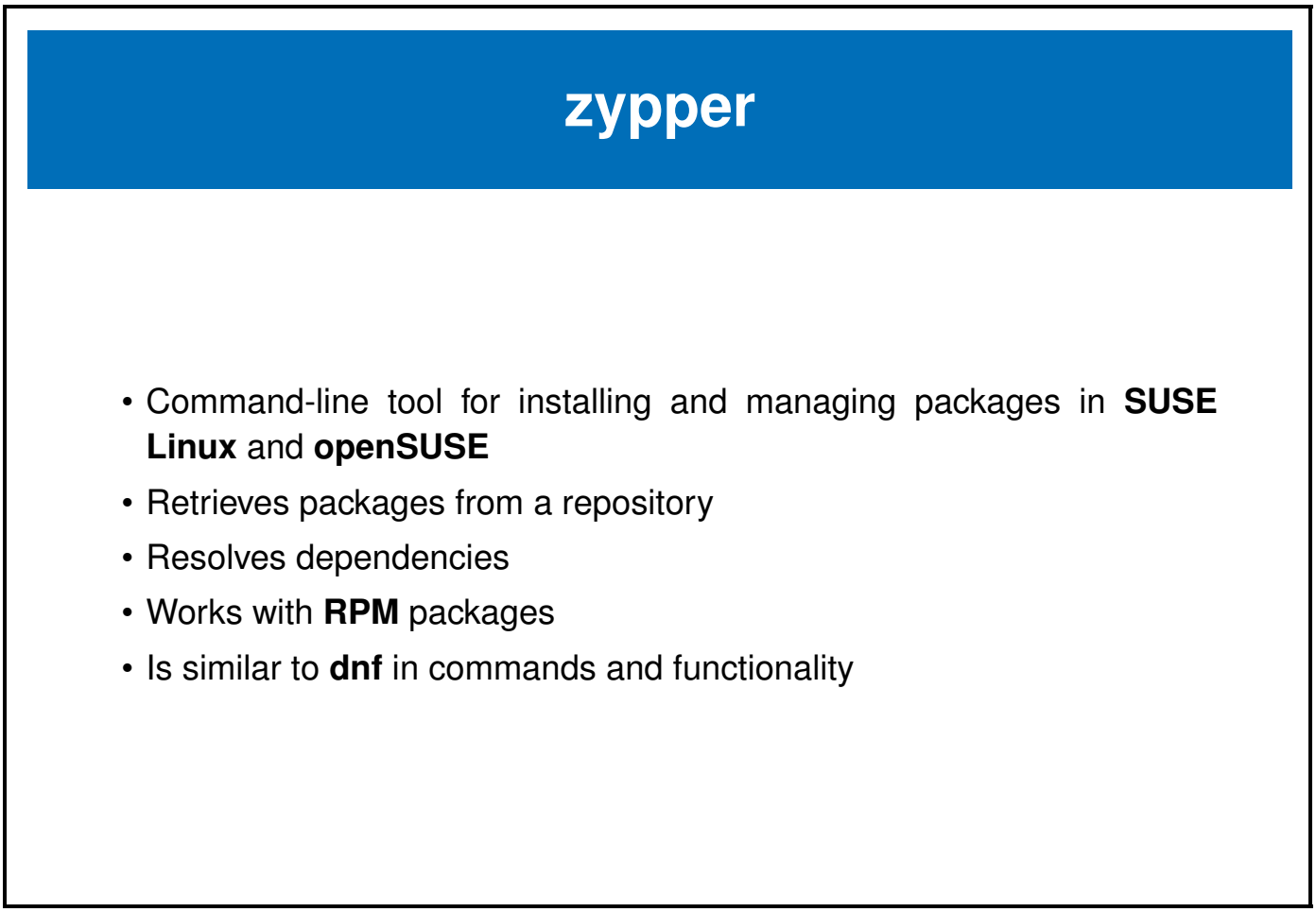
12.1 zypper 178
12.2 Queries 179
12.3 Installing/Removing/Upgrading Packages 180
12.4 Additional zypper Commands 181
12.5 Labs 182

Learning Objectives

By the end of this session, you should be able to:

- Explain what **zypper** is.
- Discuss the queries **zypper** can be used for.
- Install, remove and upgrade packages using **zypper**.
- Learn additional and more advanced **zypper** commands.

12.1 zypper

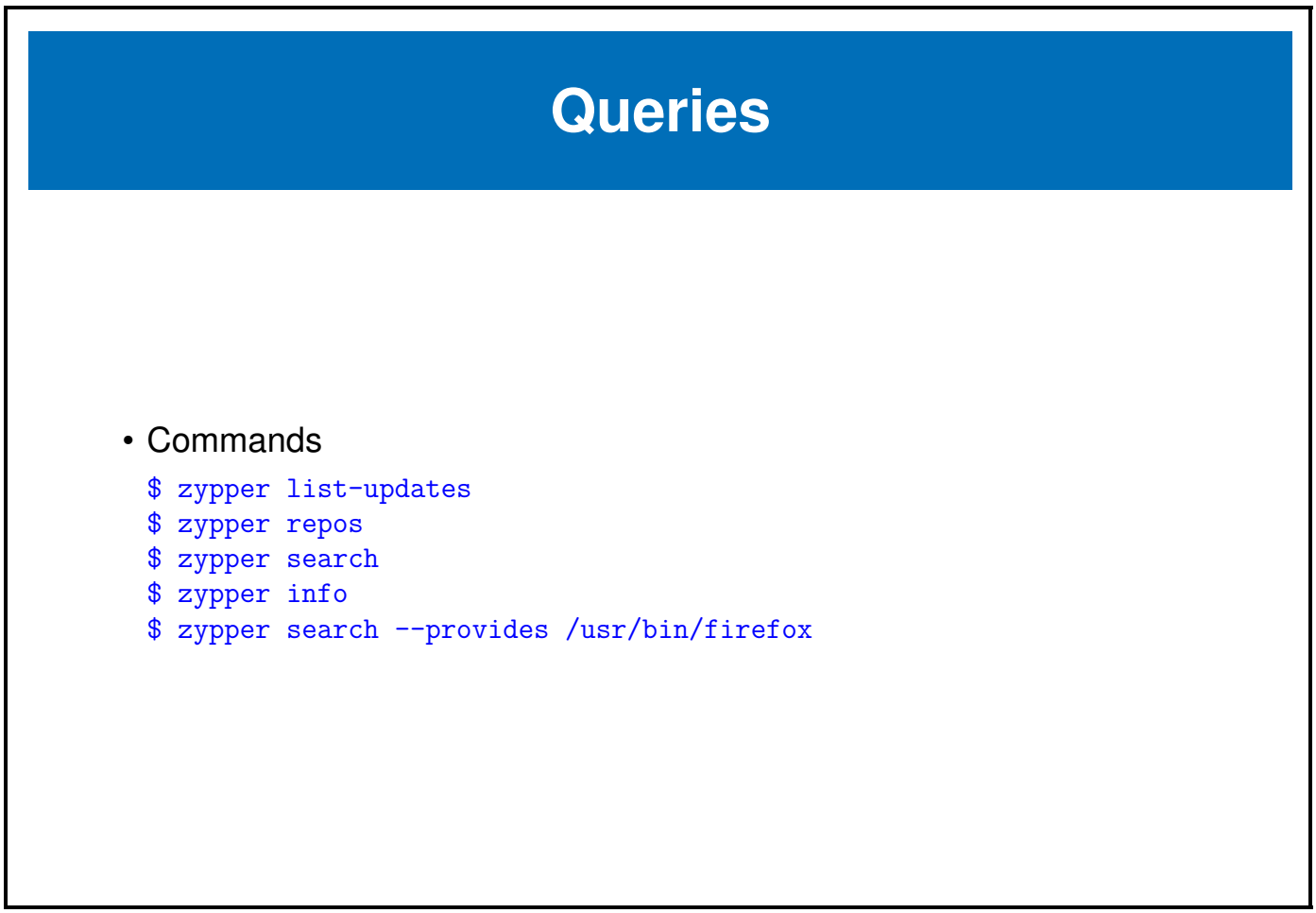


For use on **SUSE**-based systems, the **zypper** program provides a higher level of intelligent services for using the underlying **rpm** program, and plays the same role as **dnf**/**yum** on **Red Hat**-based systems. It can automatically resolve dependencies when installing, updating and removing 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.

© Copyright the Linux Foundation 2023. All rights reserved.

12.2 Queries




```
$ 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.

© Copyright the Linux Foundation 2023. All rights reserved.

12.3 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.

© Copyright the Linux Foundation 2023. All rights reserved.

12.4 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.

© Copyright the Linux Foundation 2023. All rights reserved.

12.5 Labs

Video Demonstration Resources

using_yast_demo.mp4

Exercise 12.0: 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 12.1: 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 12.1

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 12.2: 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.

© Copyright the Linux Foundation 2023. All rights reserved.
   ```

Try the commands you used above both as `root` and as a regular user. Do you notice any difference?

Solution 12.2

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.

© Copyright the Linux Foundation 2023. All rights reserved.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 13

GIT Fundamentals

We explain the concepts of all **revision control systems** and learn the first steps in using **git** - the most popular such system, especially in Open Source.

Instead of focusing on how to use **git** for developing software, we are going to focus on the multiple ways to reference a set of files (like a software package) for the purpose of installing and updating system or user software.

13.1 Revision Control 186
13.2 Basic Commands 188
13.3 Try some git commands 190
13.4 Get Software with Git 196
13.5 Labs 202

Learning Objectives

By the end of this session, you should be able to:

- Explain why revision control systems are important.
- Be familiar with the major commands used with **git** including as **clone**, **add**, **commit**, **tag**, and **checkout**.
- Get on-system help with any particular command with **git help**.
- Walk through and example that can be adapted to using **git** for simple, linear tracking of system configuration files.
- Learn about how to download **upstream** versions of software, including the typical use of **tags** and **branches**.

13.1 Revision Control

What is a Revision Control System

git is a **Revision Control System** . . .

A **Revision Control System** is used to track changes by storing versions of the file at a point in time capturing:

- When – a timestamp of the point in time
- What – a version of the file at the point in time
- Who – what user was associated with recording the change
- Why – what extra information would be useful to know about the change

Even very early systems like **RCS** tracked this information

In this early system, the changes were tracked per file in a text file with the same name but with an appended **,v** in a subdirectory **RCS/**. This system lacked the ability to group a change across multiple files and had drawbacks in file locking and storage efficiency, but the basic precepts above still made this a powerful tool.

© Copyright the Linux Foundation 2023. All rights reserved.

What is a Revision Control System

git is a **Distributed** Revision Control System...

The next step in the evolution from our one file, one change system was to allow multiple changes to be tracked at the same time and for the **source of truth** to be a centralized network server. **CVS** used the same underlying mechanism for storing file changesets, but allowed developers with access to the same network to work together. Unfortunately, problems with exclusive access and changes applying to the same files made this system a hardship at times.

A **Distributed** Revision Control System allows for many contributors (thousands in the case of the **Linux** kernel)

- Many projects have developers working separately, united by a common version control system. This is possible even when there is a central authoritative repository, such as with **CVS**.
- What makes **git** different is that on a technical level, there is no such thing as central authoritative repository; the underlying framework is actually **peer-to-peer** in nature. The importance and central role played by one particular location is political and sociological, not technical.
- **Distributed development** is not defined by various parts of a project being worked on separately, with a partition between different host repositories. In **git**, distributed development means every repository is authoritative, and contains not just one part of the project, but the entire code base.
- **git** has no politics, and has no preferred model for organization. The hierarchy of the development community can be very centralized and top down, or it can be very flat. The exchange of information and changes can be pyramidal, with a number of development trees coalescing into a top level one, or it can be very egalitarian.
- **git** is a tool, not a rigid method, and it can be used in any way that fits the needs and philosophy of a project.

© Copyright the Linux Foundation 2023. All rights reserved.

13.2 Basic Commands

Some Essential Commands

```
$ git --version
$ git help [subcommand]
$ git clone
$ git branch
$ git status
$ git add
$ git commit
$ git merge
$ git pull
$ git push
```

You can see the version of **git** you have installed with:

```
$ git --version
```

```
git version 2.30.2
```

Built-in Help

The **git help** subcommand provides detailed man pages for each git invocation.

```
$ git help
$ git help command
```


© Copyright the Linux Foundation 2023. All rights reserved.

Core git commands

```
start a working area (see also: git help tutorial)
   clone             Clone a repository into a new directory
   init              Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add               Add file contents to the index
   mv                Move or rename a file, a directory, or a symlink
   restore           Restore working tree files
   rm                Remove files from the working tree and from the index
   sparse-checkout   Initialize and modify the sparse-checkout

examine the history and state (see also: git help revisions)
   bisect            Use binary search to find the commit that introduced a bug
   diff              Show changes between commits, commit and working tree, etc
   grep              Print lines matching a pattern
   log               Show commit logs
   show              Show various types of objects
   status            Show the working tree status

grow, mark and tweak your common history
   branch            List, create, or delete branches
   commit            Record changes to the repository
   merge             Join two or more development histories together
   rebase            Reapply commits on top of another base tip
   reset             Reset current HEAD to the specified state
   switch            Switch branches
   tag               Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch             Download objects and refs from another repository
   pull              Fetch from and integrate with another repository or a local branch
   push              Update remote refs along with associated objects
```

You can get a basic list of **git** commands by just typing `git`, which will give the list pictured above.

There are only a few global options that apply, those prefixed with `--` in the above listing. Many of the subcommands have their own options which are included in `[ARGS]` in the above.

Want More?

Visit the official reference where you can find the complete list of subcommands, envrionment variables, and more - by specific git version:
https://git-scm.com/docs/git#_git_commands

© Copyright the Linux Foundation 2023. All rights reserved.

13.3 Try some git commands

Minimal Global Config

Unlike some other recent revision control systems, git wants you to set up an **author** `name` and `email` address rather than just default to your current userid.

Let's set that up now and additionally prefer the more modern default branch name *main*.

```
$ git config --global user.name "Gladys West"
$ git config --global user.email "gwest@example.com"
$ git config --global init.defaultBranch main
```

These settings will get written to your `~/.gitconfig`.

The `~/.gitconfig` file can contain quite a lot more. Many people have shortcuts for frequently typed commands, custom ways to print histories, and different **diff** commands.

© Copyright the Linux Foundation 2023. All rights reserved.

Create and Initialize Repository

Create a new directory and initialize it as a git repository.

```
$ mkdir /tmp/LFgit
$ cd /tmp/LFgit
$ git init
```

```
Initialized empty Git repository in /tmp/LFgit/.git/
```

```
$ git status
```

```
On branch main

No commits yet

nothing to commit (create/copy files and use "git add'' to track)
```

Let's inspect what happened so far:

```
$ ls -la /tmp/LFgit
```

```
total 12
drwxr-xr-x  3 user user 4096 May  9 18:12 .
drwxrwxrwt 16 root root 4096 May  9 18:16 ..
drwxr-xr-x  7 user user 4096 May  9 18:15 .git
```

```
$ ls -la /tmp/LFgit/.git
```

```
ls -l /tmp/LFgit/.git/
total 32
drwxr-xr-x 2 user user 4096 May  9 18:12 branches
-rw-r--r-- 1 user user   92 May  9 18:12 config
-rw-r--r-- 1 user user   73 May  9 18:12 description
-rw-r--r-- 1 user user   21 May  9 18:12 HEAD
drwxr-xr-x 2 user user 4096 May  9 18:12 hooks
drwxr-xr-x 2 user user 4096 May  9 18:12 info
drwxr-xr-x 4 user user 4096 May  9 18:12 objects
drwxr-xr-x 4 user user 4096 May  9 18:12 refs
```


© Copyright the Linux Foundation 2023. All rights reserved.

Add a file to the repository

Copy /etc/motd into our repository – it's small and (hopefully) only contains text.

```
# (still in /tmp/LFgit)
$ cp /etc/motd .
$ git status -sb -uall
```

```
## No commits yet on main
?? motd
```

```
$ git add motd
$ git status -sb -uall
```

```
## No commits yet on main
A  motd
```

Now we see that our file has been **A**dded to the staging area.

Notice that we used a slightly different invocation of **git status**

- **-s** display short format
- **-b** show branch information even in short format
- **-uall** show untracked files including in untracked directories

© Copyright the Linux Foundation 2023. All rights reserved.

Change it Up

Not we are going to modify our file that was adding to the staging area but not committed yet. Does it matter?

```
$ echo meow >> motd
$ git status -sb -uall
```

```
## No commits yet on main
AM motd
```

Our status has updated, our file is still **A**dded but now also **M**odifiifed. What's the modification?

```
$ git diff
```

```
diff --git a/motd b/motd
index 0c87dd3..c3ce8b3 100644
--- a/motd
+++ b/motd
@@ -5,3 +5,4 @@ individual files in /usr/share/doc/*/copyright.

 Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
 permitted by applicable law.
+meow
```

We were able to add a file, modify it, and see the differences even though there hasn't been a single commit yet.

Not every modern revision control system behaves this way. For example, *mercurial (hg)* will not treat a staging area in this fashion.

© Copyright the Linux Foundation 2023. All rights reserved.

Commit It

Commit the file and check our status. Which version of the file do you think will be committed?

```
$ git commit -m 'message of the day'
$ git status -sb -uall
```

```
## main
 M motd
```

It committed the version of the file that was added into our staging area. If we really wanted the current version of the file with the extra "meow", then we need to issue another **git add** and another **git commit**.

Or we could also do this in one step without creating another commit with **git commit --all --amend**

--all automatically stage modified and deleted (not new) files

--amend replace the last commit with a new commit

```
$ git commit --all --amend
```

commit msg

```
add critical 'meow' to motd

    * Closes #4563: Where did meow go?

    * After serious testing, we've discovered a missing word that is
      causing production issues.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
```

Correctly formatted commit messages can go a long way in providing a professional appearance to your work. When you create a commit message, here are a few rules to follow:

- the top summary line is short - keep it to 50 characters or so
- use current tense – *what will the commit do when applied*
- add more detail in subsequent lines - skip a line and indent

© Copyright the Linux Foundation 2023. All rights reserved.

Check Our State

What is our current situation?

```
$ git status
```

```
On branch main
nothing to commit, working tree clean
```

```
$ git log
```

```
commit 8933f422d17d107d35b80f8a1c55b09dbee289fe (HEAD -> main)
Author: Gladys West <gwest@example.com>
Date:   Fri May 9 20:01:53 2023 -0400

    add critical 'meow' to motd

            * Closes #4563: Where did meow go?

            * After serious testing, we've discovered a missing word that is
              causing production issues.

commit 0872249ec1c9daeb60cb491fe6be7647cf606149
Author: Gladys West <gwest@example.com>
Date:   Fri May 9 19:31:03 2023 -0400

    message of the day
```

So we have a clean working directory and both of our commits are showing in the log.

The **git log** command has a large amount of configuration available – including relative times, diffs of files changed, and extra fields that are not normally displayed.

Here's a version with a **short commit hash** and fitting on one line:

```
git  log --pretty=oneline -n 20 --graph --abbrev-commit
```

```
* 8933f42 (HEAD -> main) add critical 'meow' to motd
* 0872249 message of the day
```


© Copyright the Linux Foundation 2023. All rights reserved.

13.4 Get Software with Git

Download Software

There are many instances where you are not interested in modifying and tracking changes to files at all, you just want to download the software package or config, and you are using git because:

- That's the author's recommended way to get this software
- It's the newer or more complete than the distribution version
- It's a fast way to look at the source and the changes

Entire ecosystems or **software forges** have developed around git software development and can be used with just a web browser. Some popular ones include:

- **GitHub**
- **GitLab**
- **Codeberg**
- an author or organization *self-hosted* software forge

Usually only specific release versions are readily downloadable using only a web browser, but taking part in discussions, filing issues, and browsing course code is often possible even from a mobile browser.

You can look at random projects this way too. Each of the specific sites above supports adding `"/explore/"` to the end of the hostname:

- https://github.com/explore/
- https://gitlab.com/explore/
- https://codeberg.org/explore/

© Copyright the Linux Foundation 2023. All rights reserved.

Get the Linux Kernel

Let's use the **clone** subcommand to our own local copy of the **kernel** source.

```
$ git clone --depth 1 -b master \
  https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git linux/
```

```
Cloning into '/home/user/linux'...
remote: Enumerating objects: 85093, done.
remote: Counting objects: 100% (85093/85093), done.
remote: Compressing objects: 100% (82795/82795), done.
remote: Total 85093 (delta 6421), reused 17237 (delta 1404), pack-reused 0
Receiving objects: 100% (85093/85093), 237.53 MiB | 1.34 MiB/s, done.
Resolving deltas: 100% (6421/6421), done.
Updating files: 100% (80335/80335), done.
```

For large projects, if you know you just want to look at the very latest (or another) commit without all of the history, in exchange for less time downloading and less disk space used, you can specify some additional options.

Without these options, **git clone** would retrieve the **default branch** with all of the history.

We passed the following options to **git clone**

- **-b** point to the named branch when cloning
- **-depth** create a shallow clone with a specified number of commits

Let's check the results of our clone command.

```
$ git log
```

```
commit 9561de3a55bed6bdd44a12820ba81ec416e705a7 (grafted, HEAD -> master, tag: v6.4-rc5, origin/master, origin/HEAD)
Author: Linus Torvalds <torvalds@linux-foundation.org>
Date:   Sun Jun 4 14:04:27 2023 -0400

    Linux 6.4-rc5
```

Note that there was just one entry present in the commit log.

Want more on the Linux kernel?

The Linux Foundation has a range of classes covering a variety of kernel development topics, including embedded systems and kernel security:
https://training.linuxfoundation.org/full-catalog/?_sft_topic_area=linux-kernel-development

© Copyright the Linux Foundation 2023. All rights reserved.

Treeish

We specified a **branch** when we downloaded the latest **Linux** kernel just now. What does that mean?

- Most git commands operate on a **commit reference** – a specific collection of files at a specific moment in time.
- A **treeish** is the syntax to specify a **commit reference**, including a:
  - full **commit hash** – `9561de3a55bed6bdd44a12820ba81ec416e705a7`
  - shorter version of that hash – `9561de3a5`
  - **branch** – `main` or `devel`
  - **tag** – `v6.4-rc5`
  - a specific **remote** version – `refs/remotes/origin/main`

Often the `README` or `INSTALL` file in a project directory will specify how best to retrieve specific versions of that project. It is important to check this information because the naming and use of **git** features and those of the **code forge** will vary greatly between projects.

For example, sometimes `main` (or `master` for older projects) will refer to the unstable, working branch, but other times it will refer to the version that has been tested and is suitable for a release.

Want More?

The *git-revisions* man page has more examples of using a treeish to point to a specific commit:
https://mirrors.edge.kernel.org/pub/software/scm/git/docs/gitrevisions.html#_specifying_revisions

© Copyright the Linux Foundation 2023. All rights reserved.

Branch Analogy

It can be useful to employ analogies to physical objects to understand git concepts. If we think of each commit like a page in a binder, then we can think of a **branch** as a binder that happens to have an organizer tab at the very back to make it easy to flip the the last and latest page.

When we make a **commit** by adding a page at the end of the binder, that end tab is still going to bring us quickly to the last and latest page/**commit**.

Making a new **branch** from this one is akin to copying all pages and putting them in a new binder. If we then add new pages to the new binder, our old binder is unchanged with the original last page and final tab.

Finally, let's think about each of these binder/**branches** that we've made as all residing on the same shelf, with that shelf being our git **repository**, residing locally in a directory with a `.git` subdirectory.

To create a new branch from the current branch as the source for the copy:

```
$ git checkout -b <name>
```

To create a new branch with the upstream branch as the source for the copy:

```
$ git checkout -b <name> remotes/origin/main
```


© Copyright the Linux Foundation 2023. All rights reserved.

Fork? Clone?

Continuing with the analogy, if we are envisioning our shelf as our git repository, containing a number of binders as **branches**, then we make a **clone** by copying one or more of those binder/**branches** and putting them on completely new ***physical*** shelf/**repository**.

fork and **clone** effectively mean the same thing in the context of git usage; **Github** popularized this use of the term **fork**, while **clone** is the name of the functionality inside the git program.

Very Important

Don't trust a `git clone` as a backup if it is on the same filesystem. When we make run **git clone** locally, **git** creates a copy of of the `.git` file in a new subdirectory, but to save space it uses **hard links** where possible.

The meaning of **fork** is further overloaded. Historically it meant a copy of software that was taking a diverging development path. Now it also refers to a standalone (physical) copy of a branch or branches. But it may mean a standalone copy that is **ALSO** a diverging development path. It can be confusing.

To make a local on the same filesystem without using hard links:

```
$ cd /tmp/LFgit
$ git clone . --no-hardlinks /tmp/meow/
```

```
Cloning into '/tmp/meow'...
done.
```


© Copyright the Linux Foundation 2023. All rights reserved.

Tag a Specific Spot

As the last part of our example analogy, we can think of a **tag** as another quick reference tab in our binder that *does not automatically move* with new commits.

tags are used as a way to provide an additional alias to a specific commit, often pointing to a release version.

In order to move a tag from one commit to another, it needs to be deleted and created again.

Don't use the default

Most projects use **annotated tags** or **signed tags**, which require their own commit message. The default **git tag** behavior does not ask for a commit message and these so-called *lightweight* tags are ignored by some tools.

Using **git tag** with the **sign** option requires setting up `GnuPG`, which we are not going to cover. However, except for the last step, creating an **annotated** tag is the same process:

git tag

-a, –annotated <tag name> make an unsigned annotated tag object

Going back to our example directory:

```
$ cd /tmp/LFgit
$ git tag -a v1.0 -m 'release intial version'
$ git log --pretty=oneline v1.0
```

```
8933f422d17d107d35b80f8a1c55b09dbee289fe (HEAD -> main, tag: v1.0) add critical 'meow' to motd
0872249ec1c9daeb60cb491fe6be7647cf606149 message of the day
```

v for version

It's good practice to have a different naming scheme for **branches** and **tags**. Often times a **tag** tracking release versions will start with a **v**. So a branch that is working towards the next release of version 3.17 might be simply *3.17*, while the completed release might be tagged *v3.17*.

By following this convention, you can be sure that a **branch** or a **tag** can be referenced with the simplest **treeish**.

© Copyright the Linux Foundation 2023. All rights reserved.

13.5 Labs

Exercise 13.1: Clone an upstream software repository and make local changes

We are going to walk through a relatively common task in systems management – dealing with a software package that our distribution either doesn't have packaged or its the version is out of date.

The package we will use is *moreutils* from http://joeyh.name/code/moreutils/, which **is** probably packaged by your distribution and up to date, but this example:

1. has a fair number of **branches** and **tags** to look at
2. doesn't require a large number of dependencies like many great `Rust` or `Go` utilities.
3. is actually useful if you want to install it and keep it around

We will target the **Debian** version to be a little nicer to the source server and to allow for a bit more repository information. Use this url for the exercise:

https://salsa.debian.org/nsc/moreutils

1. First **clone** the repository.
2. Create a local branch called `local/latest` based off the `debian/latest` branch.
3. Create a file called `README.local` inside the repository with anything you want.
4. Commit your new file on the new branch. Refer to the previous section **Minimum Global Config** if you need to set up a `user.name` and `user.email`
5. (Optional) Create an annotated tag based on the current version.
6. (Optional) Build the package and experiment with the tools.

At various stages of doing all this, examine the contents of the `.git` directory and see what files are being changed, what their content is, etc. Learn as much as you can.

Solution 13.1

1. `$ git clone https://salsa.debian.org/nsc/moreutils`

```
$ git clone https://salsa.debian.org/nsc/moreutils
Cloning into 'moreutils'...
warning: redirecting to https://salsa.debian.org/nsc/moreutils.git/
remote: Enumerating objects: 2107, done.
remote: Counting objects: 100% (229/229), done.
remote: Compressing objects: 100% (105/105), done.
remote: Total 2107 (delta 129), reused 208 (delta 121), pack-reused 1878
Receiving objects: 100% (2107/2107), 510.79 KiB | 1.45 MiB/s, done.
Resolving deltas: 100% (1349/1349), done.
```

2. `$ git checkout debian/latest`

```
Already on 'debian/latest'
Your branch is up to date with 'origin/debian/latest'.
```

(OR)

```
$ git status
```


© Copyright the Linux Foundation 2023. All rights reserved.

```
On branch debian/latest
Your branch is up to date with 'origin/debian/latest'.

nothing to commit, working tree clean
```

This was actually a **NOOP** because the **default branch** is already `debian/latest`.

So either of these commands will work:

```
$ git checkout -b local/latest
```

```
Switched to a new branch 'local/latest'
```

(OR)

```
$ git checkout -b local/latest debian/latest
```

```
Switched to a new branch 'local/latest'
```

But the last one will work regardless of which textbfbranch you currently have checked out.

3. `$ touch README.local`

4.
```
$ git add README.local
$ git commit README.local
```

```
[local/latest cc7d808] add local Tracking file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.local
```

5. `$ git tag -a vlocal/1.0 -m 'first local version'`

© Copyright the Linux Foundation 2023. All rights reserved.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 14

Processes

14.1 Programs and Processes . . . 206
14.2 Process Limits . . . 210
14.3 Creating Processes . . . 213
14.4 Process Control . . . 215
14.5 Starting Processes in the Future . . . 218
14.6 Process States . . . 222
14.7 Execution Modes . . . 223
14.8 Daemons . . . 226
14.9 niceness . . . 227
14.10 Labs . . . 229

Learning Objectives

By the end of this session, you should be able to:

- Describe a process and the resources associated with it.
- Distinguish between processes, programs and threads.
- Understand process attributes, permissions and states, and know how to control limits.
- Talk about how processes are created, and how they can be put in background and brought back to foreground.
- Manage **jobs** (background processes), use **at** to start processes in the future, **sleep** to pause execution, and **cron** to schedule periodic execution.
- Explain possible process states.
- Explain the difference between running in user and kernel modes.
- Describe daemon processes.
- Understand how new processes are forked (created).
- Use nice and renice to set and modify priorities.

14.1 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

One should 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).

© Copyright the Linux Foundation 2023. All rights reserved.

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`.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

14.2 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.

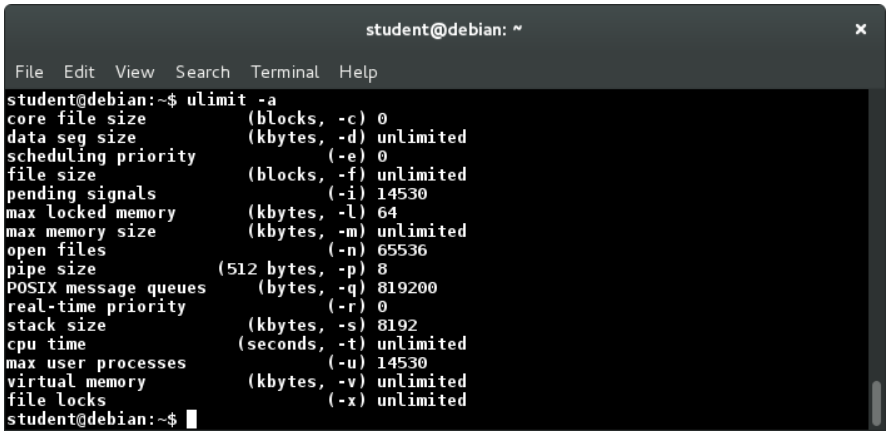


Figure 14.1: **ulimit**

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

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
```

```
4096
```

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

```
1024
```


© Copyright the Linux Foundation 2023. All rights reserved.

14.3 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.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

14.4 Process Control

Background and Foreground Processes

- Processes run in **foreground** by default
- Adding an ampersand (&) after a command will run the command in the background:

```
$ sudo updatedb &
```

- CTRL-Z suspends a foreground process
  - **bg** makes it run in the background
  - **fg** puts it back in the foreground
- CTRL-C terminates a foreground process

© Copyright the Linux Foundation 2023. All rights reserved.

Foreground jobs run from the shell, delaying access to the shell wait until the job has finished. This may not be a problem, but if the job will take a long time to complete, putting it in the background frees up the shell for further interactive work. The background job will run at a lower priority allowing interactive work to go smoothly. You can also log off the terminal window without affecting the background job.

```
 $ updatedb &
[1] 22209
$ sleep 100
^Z
```

```
[2]+  Stopped                 sleep 100
```

```
$ bg
```

```
[2]+ sleep 100 &
```

```
$ fg
```

```
sleep 100
^Z
[2]+  Stopped                 sleep 100
```

```
$ bg
```

```
[2]+ sleep 100 &
```

```
$ fg
```

```
sleep 100
^C
```


© Copyright the Linux Foundation 2023. All rights reserved.

Managing Jobs

- **jobs** shows background processes in the current terminal
- Shows job ID, state and command name:

```
$ jobs
```

```
1 -  Running     updatedb &
2 +  Stopped     sleep 10
```

- Job ID's can be used with **bg** and **fg**

```
$ jobs -l
```

will provide the PID for the job.

The background jobs are still somewhat connected to the terminal window, in that, if you log off, **jobs** will no longer show the jobs started from that window.

© Copyright the Linux Foundation 2023. All rights reserved.

14.5 Starting Processes in the Future

Using at to Start in the Future

- Executes any non-interactive command at a specified time
- Start command after a delay:

```
$ at now + 2 days
at> mail < /var/log/messages admin@example.com
at> <EOT>
job 1 at 2013-01-16 13:24
```

- Use CTRL-D to insert the EOT character
- Run **atq** for examining or deleting queued jobs

By entering **at** with the future time, the interactive part of **at** will start. At the prompt, enter in the command to run, hit enter, then CTRL-D to exit. You then use **atq** to see the job information.

© Copyright the Linux Foundation 2023. All rights reserved.

cron

- **cron** is used to schedule commands at specific intervals
- Can be managed in different ways:
  - **crontab** lets users specify jobs
  - `/etc/cron.d/` can be extended with formatted job files
  - `/etc/cron.{hourly,daily,weekly,monthly}` can contain any system script
- While **cron** has been used in **UNIX**-like operating systems for decades, modern **Linux** distributions have moved over to **anacron** as will be explained shortly.

© Copyright the Linux Foundation 2023. All rights reserved.

cron is a very useful and flexible tool. It is used for any job that needs to run on a regular schedule; e.g. backups.

Tasks can be queued to run every hour, every day, once a week or once a month, or even every 10 seconds. Mail will be sent when a job has completed or failed.

```
$ ls -l /etc/cron.d
```

```
total 16
-rw-r--r-- 1 root root 128 Mar 29  2017 0hourly
-rw-r--r-- 1 root root  78 Dec 29 09:24 atop
-rw-r--r-- 1 root root 108 Jun 13  2017 raid-check
-rw------- 1 root root 235 Mar 29  2017 sysstat
```

```
$ ls -l /etc/cron.daily
```

```
total 68
-rwxr-xr-x 1 root root   302 Mar  7 08:55 docker-latest-logrotate
-rwxr-xr-x 1 root root 18812 Mar 20 00:01 google-chrome
-rwxr-xr-x 1 root root     1 Jan 17  2017 google-earth
lrwxrwxrwx 1 root root    45 Jun 16  2016 google-talkplugin ->
↪  /opt/google/talkplugin/cron/google-talkplugin
-rwx------ 1 root root   219 Jan 24  2017 logrotate
-rwxr-xr-x 1 root root   618 Mar 17  2014 man-db.cron
-rwx------ 1 root root   208 Feb  4  2016 mlocate
-rwxr-x--- 1 root root  2126 Aug 17  2016 prelink
-rwx------ 1 root root   256 Sep  1  2017 rhsmd
-rwxrwxr-x 1 root root 18775 Mar  5 13:55 slack
```

anacron

- On older **Linux** systems, `crontab` contained information about when to run the jobs in the `/etc/cron.*` subdirectories
- However, it was implicitly assumed the machine was always running.
- If the machine was powered off, scheduled jobs would not run.
- **anacron** has replaced **cron** on modern systems
- **anacron** will run the necessary jobs in a controlled and staggered manner when the system is running.
- The key configuration file is `/etc/anacrontab`

© Copyright the Linux Foundation 2023. All rights reserved.

example **anacron** configuration file:

/etc/anacrontab

```
# /etc/anacrontab: configuration file for anacron
# See anacron(8) and anacrontab(5) for details.

SHELL=/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
# the maximal random delay added to the base delay of the jobs
RANDOM_DELAY=45
# the jobs will be started during the following hours only
START_HOURS_RANGE=3-22

#period in days   delay in minutes   job-identifier   command
1       5       cron.daily              nice run-parts /etc/cron.daily
7       25      cron.weekly             nice run-parts /etc/cron.weekly
```

Sleeping

- **sleep** can be used to easily delay commands:

```
$ (sleep 600; launchbackup.sh) &
```

sleep can be used in shell scripts as well – for example, as a reminder every 10 minutes that a meeting time is approaching. The argument defaults to *seconds*, but you can explicitly set *s* for *seconds* for clarify, *m* for *minutes*, *h* for *hours* or *d* for *days*.

© Copyright the Linux Foundation 2023. All rights reserved.

14.6 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.

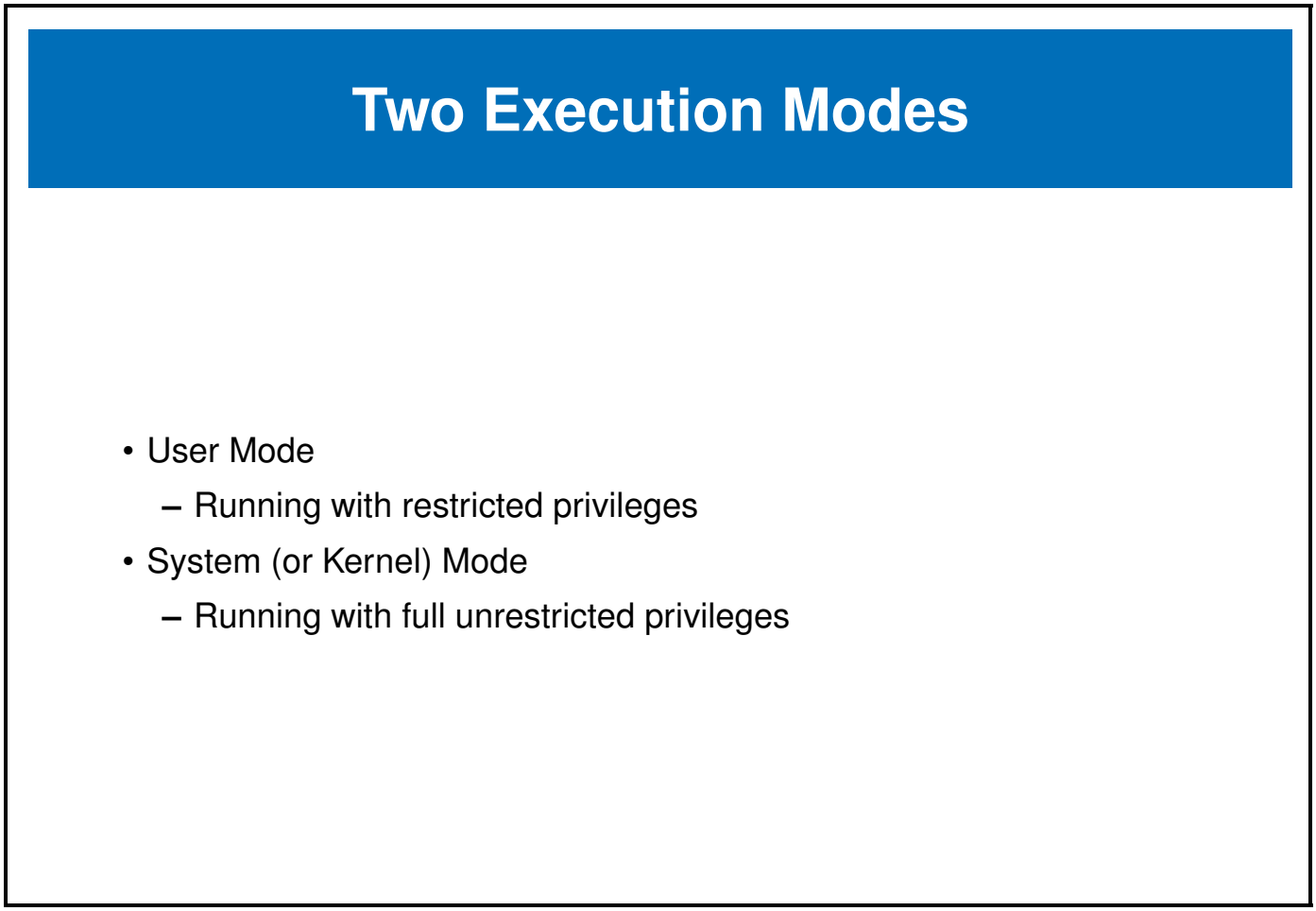
A process may be **Stopped**, meaning execution of instructions has been suspended. This state is commonly experienced when a programmer wants to examine the executing programs memory, CPU registers, flags or other attributes. Once this is done the process may be resumed.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

14.7 Execution Modes



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**.

© Copyright the Linux Foundation 2023. All rights reserved.

User Mode

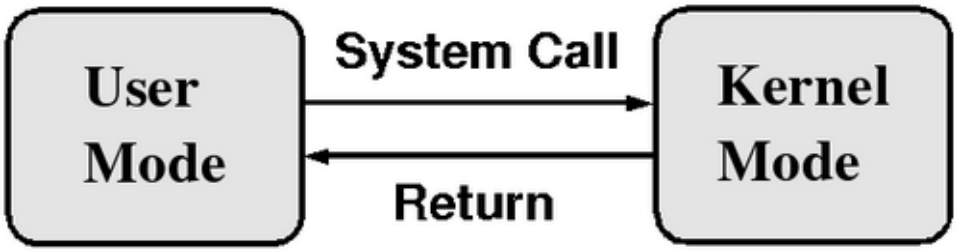


Figure 14.2: **User and Kernel Modes and System Calls**

- **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.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

14.8 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. It's sole purpose on the system is to provide that specific service. Some examples include **xinetd**, **httpd**, **lpd**, and **vsftpd**.

© Copyright the Linux Foundation 2023. All rights reserved.

14.9 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.

© Copyright the Linux Foundation 2023. All rights reserved.

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**.

© Copyright the Linux Foundation 2023. All rights reserved.

14.10 Labs

Video Demonstration Resources

using_renice_demo.mp4

Exercise 14.1: 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 14.1

1.
```
$ bash
$ ulimit -n
```
```
1024
```
```
$ ulimit -S -n
```
```
1024
```
```
$ ulimit -H -n
```
```
4096
```

2.
```
$ ulimit -n hard
$ ulimit -n
```
```
4096
```

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

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


© Copyright the Linux Foundation 2023. All rights reserved.

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

```
$ ulimit -n
```

```
2048
```

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 14.2: Examining System V IPC Activity

System V IPC is a rather old method of **I**nter **P**rocess **C**ommunication 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
```

```
------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x01114703 0          root       600        1000       6
0x00000000 98305      coop       600        4194304    2          dest
0x00000000 196610     coop       600        4194304    2          dest
0x00000000 23068675   coop       700        1138176    2          dest
0x00000000 23101444   coop       600        393216     2          dest
0x00000000 23134213   coop       600        524288     2          dest
0x00000000 24051718   coop       600        393216     2          dest
0x00000000 23756807   coop       600        524288     2          dest
0x00000000 24018952   coop       600        67108864   2          dest
0x00000000 23363593   coop       700        95408      2          dest
0x00000000 1441811    coop       600        2097152    2          dest

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0x00000000 98304      apache     600        1
0x00000000 131073     apache     600        1
0x00000000 163842     apache     600        1
0x00000000 196611     apache     600        1
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:

© Copyright the Linux Foundation 2023. All rights reserved.

```
$ ipcs -p
```

```
------ Message Queues PIDs --------
msqid      owner      lspid      lrpid

------ Shared Memory Creator/Last-op PIDs --------
shmid      owner      cpid       lpid
0          root       1023       1023
98305      coop       2265       18780
196610     coop       2138       18775
23068675   coop       989        1663
23101444   coop       989        1663
23134213   coop       989        1663
24051718   coop       20573      1663
23756807   coop       10735      1663
24018952   coop       17875      1663
23363593   coop       989        1663
1441811    coop       2048       20573
```

Thus, by doing:

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

```
coop      2048  5.3  3.7 1922996 305660 ?      Rl   Oct27  77:07 /usr/bin/gnome-shell
coop     20573  1.9  1.7 807944 141688 ?       Sl   09:56   0:01 /usr/lib64/thunderbird/thunderbird
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
```

```
....
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
....
0x00000000 622601     coop       600        2097152    2          dest
0x0000001a 13303818   coop       666        8196       0
....
```

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.

Exercise 14.3: Getting Uptime and Load Average

Ascertain how long your system has been up, and also display its load average.

Solution 14.3

A very simple method is just to use the **uptime** utility:

```
$ uptime
```

```
10:26:40 up  3:19,  5 users,  load average: 1.46, 1.40, 1.19
```


© Copyright the Linux Foundation 2023. All rights reserved.

A second method is to look at the first line of output from **top**:

```
$ top | head
```

```
top - 10:28:11 up  3:20,  5 users,  load average: 1.93, 1.52, 1.25
Tasks: 313 total,   1 running, 312 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.0 us,  0.3 sy,  0.0 ni, 98.2 id,  0.5 wa,  0.0 hi,  0.0 si,  0.0
KiB Mem : 16284472 total,  6556792 free,  1029760 used,  8697920 buff/cache
KiB Swap:  8290300 total,  8290300 free,        0 used. 10364220 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 2615 coop      20   0  504536 186312  65488 S   6.7  1.1   6:28.30 skype-b+
18248 coop      20   0  655804  50816  30884 S   6.7  0.3   0:20.11 emacs
    1 root      20   0  204912   6508   3956 S   0.0  0.0   0:00.92 systemd
```

A third method is to use **w**:

```
 10:30:51 up  3:23,  5 users,  load average: 0.55, 1.11, 1.14
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
coop     :0       :0               07:08   ?xdm?  16:51   0.19s gdm-session-
coop     pts/0    :0               07:09    2:22m  0.12s  0.12s bash
coop     pts/1    :0               07:09    1:37m  0.42s  0.42s bash
coop     pts/2    :0               07:09    0.00s 51.09s  0.00s w
coop     pts/3    :0               07:09   27:08   0.25s  0.25s bash
```

Exercise 14.4: Background and Foreground Jobs

We are going to launch a graphical program from a terminal window, so that one can no longer type in the window. **gedit** is an easy choice but you can substitute any other program that does this.

1. Open **gedit** on a new file as in

   ```
   $ gedit somefile
   ```

   You can no longer type in the terminal window.

2. While your pointer is over the terminal window, hit `CTRL-Z`.

   ```
   ^Z
   ```

   ```
   [3]+  Stopped                 gedit somefile
   ```

   You can no longer type in the **gedit** window.

3. With **jobs -l**, see what processes have been launched from this terminal window:

   ```
   $ jobs -l
   ```

   ```
   [1]  17705 Running                 evince *pdf &
   [2]- 18248 Running                 emacs /tmp/hello.tex &
   [3]+ 19827 Stopped                 gedit somefile
   ```

4. Now put the most recent job (**gedit somefile**) in background:

   ```
   $ bg
   ```


© Copyright the Linux Foundation 2023. All rights reserved.

```
[3]+ gedit somefile &
```

Now you should be able to type in the **gedit** window.

5. Put the process in foreground again:

```
$ fg
gedit somefile
```

Note you can no longer type in the terminal window.

6. To clean up, suspend the process again and then use **kill** to terminate it:

```
^Z
```

```
[3]+  Stopped                 gedit somefile
```

```
$ jobs -l
```

```
[1]  17705 Running                 evince *pdf &
[2]- 18248 Running                 emacs /tmp/hello.tex &
[3]+ 19827 Stopped                 gedit somefile
```

```
$ kill -9 19827
$ jobs -l
```

```
[1]  17705 Running                 evince *pdf &
[2]- 18248 Running                 emacs /tmp/hello.tex &
[3]+ 19827 Killed                  gedit somefile
```

```
$ jobs -l
```

```
[1]- 17705 Running                 evince *pdf &
[2]- 18248 Running                 emacs /tmp/hello.tex &
```

Exercise 14.5: Using at for Batch Processing in the Future

Schedule a very simple task to run at a future time from now. This can be as simple as running **ls** or **date** and saving the output. (You can use a time as short as one minute in the future.)

Note that the command will run in the directory from which you schedule it with **at**.

Do this:

1. From a short **bash** script.
2. Interactively

Solution 14.5

1. Create the file `testat.sh` containing:

```
#!/bin/bash
date > /tmp/datestamp
```

and then make it executable and queue it up with **at**:

© Copyright the Linux Foundation 2023. All rights reserved.

```
$ chmod +x testat.sh
$ at now + 1 minute -f testat.sh
```

You can see if the job is queued up to run with **atq**:

```
$ atq
```

```
17        Wed Apr 22 08:55:00 2015 a student
```

Make sure the job actually ran:

```
$ cat /tmp/datestamp
```

```
Wed Apr 22 08:55:00 CDT 2015
```

What happens if you take the >/tmp/datestamp out of the command? (Hint: type **mail** if not prompted to do so!)

2. Interactively it is basically the same procedure. Just queue up the job with:

```
$ at now + 1 minute
at> date > /tmp/datestamp
CTRL-D
$ atq
```

Exercise 14.6: Scheduling a Periodic Task with cron

Set up a **cron** job to do some simple task every day at 10 AM.

Solution 14.6

Create a file named `mycrontab` with the following content:

```
0 10 * * * /tmp/myjob.sh
```

and then create `/tmp/myjob.sh` containing:

SH /tmp/myjob.sh

```
#!/bin/bash
echo Hello I am running $0 at $(date)
```

and make it executable:

```
$ chmod +x /tmp/myjob.sh
```

Put it in the **crontab** system with:

```
$ crontab mycrontab
```

and verify it was loaded with:

```
$ crontab -l
```

```
0 10 * * * /tmp/myjob.sh
```

```
$ sudo ls -l /var/spool/cron/student
```


© Copyright the Linux Foundation 2023. All rights reserved.

```
-rw------- 1 student student 25 Apr 22 09:59 /var/spool/cron/student
```

```
$ sudo cat /var/spool/cron/student
0 10 * * * /tmp/myjob.sh
```

Note if you don't really want this running every day, printing out messages like:

```
Hello I am running /tmp/myjob.sh at Wed Apr 22 10:03:48 CDT 2015
```

and mailing them to you, you can remove it with:

```
$ crontab -r
```

If the machine is not up at 10 AM on a given day, **anacron** will run the job at a suitable time.

© Copyright the Linux Foundation 2023. All rights reserved.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 15

Process Monitoring

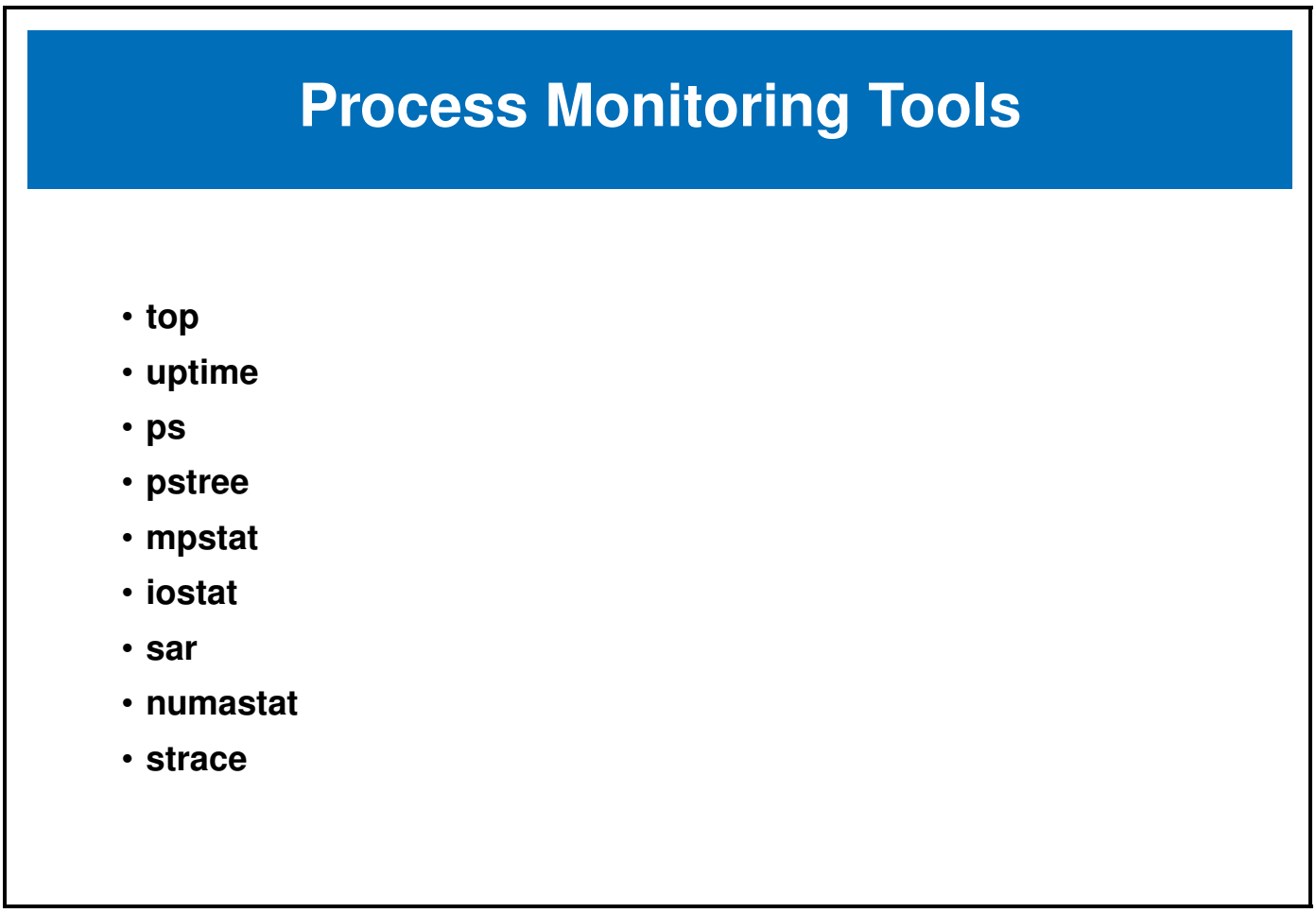
15.1 Process Monitoring 238
15.2 Troubleshooting 239
15.3 ps 241
15.4 pstree 243
15.5 top 245
15.6 Labs 247

Learning Objectives

By the end of this session, you should be able to:

- Identify and use a range of command line process monitoring tools.
- Begin troubleshooting system problems.
- Learn how to Use **ps** to view characteristics and statistics associated with processes.
- Identify different **ps** output fields and customize the ps output.
- Use **pstree** to get a visual description of the process ancestry and multi-threaded applications.
- Use **top** to view system loads interactively.

15.1 Process Monitoring



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 15.1: **Process and Load Monitoring Tools**

Tool	Purpose
top	Process activity, dynamically updated
uptime	How long the system is running and the average load
ps	Detailed information about processes
pstree	A tree of processes and their connections
mpstat	Multiple processor usage
iostat	CPU utilization and I/O statistics
sar	Display and collect information about system activity
numastat	Information about **NUMA** (Non-Uniform Memory Architecture)
strace	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.

© Copyright the Linux Foundation 2023. All rights reserved.

15.2 Troubleshooting

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.

© Copyright the Linux Foundation 2023. All rights reserved.

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 at 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.

© Copyright the Linux Foundation 2023. All rights reserved.

15.3 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 ?        Ssl  06:53   0:11  \_ /usr/libexec/gnome-terminal-server
coop        3103  0.0  0.0 237040  5648 pts/0    Ss   06:53   0:00  |   \_ bash
coop       51808  0.0  0.0 248280  6272 pts/0    S+   13:41   0:00  |   |   \_ git log
coop       51809  0.0  0.0 219456  2496 pts/0    S+   13:41   0:00  |   |       \_ /usr/bin/less
coop        3158  0.0  0.0 237512  5980 pts/1    Ss+  06:53   0:00  |   \_ bash
c8:/tmp>
```

Figure 15.1: **Using ps With BSD Options**

© Copyright the Linux Foundation 2023. All rights reserved.

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 15.2: **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 15.3: **Using ps With UNIX Options**

© Copyright the Linux Foundation 2023. All rights reserved.

15.4 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
```


© Copyright the Linux Foundation 2023. All rights reserved.

pstree gives a visual description of the process ancestry and multi-threaded applications:

```
$ pstree -aAp 2408
```

```
bash,2408
|-emacs,24998 pmonitor.tex
|   |-{emacs},25002
|   `-{emacs},25003
|-evince,18036 LFS201-SLIDES.pdf
|   |-{evince},18040
|   |-{evince},18046
|   `-{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
```

```
total 0
dr-xr-xr-x 5 coop coop 0 Sep 11 07:15 18036
dr-xr-xr-x 5 coop coop 0 Sep 11 07:15 18040
dr-xr-xr-x 5 coop coop 0 Sep 11 07:15 18046
dr-xr-xr-x 5 coop coop 0 Sep 11 07:15 18047
```


© Copyright the Linux Foundation 2023. All rights reserved.

15.5 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`).

```
top - 09:42:28 up  2:56,  1 user,  load average: 1.20, 1.16, 1.09
Tasks: 339 total,   1 running, 338 sleeping,   0 stopped,   0 zombie
%Cpu0  :  1.0 us,  0.3 sy,  0.0 ni, 98.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  1.3 us,  0.3 sy,  0.0 ni, 98.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.7 us,  0.3 sy,  0.0 ni, 99.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  0.7 us,  0.3 sy,  0.0 ni, 99.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :  0.3 us,  0.3 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :  2.0 us,  0.7 sy,  0.0 ni, 97.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :  1.7 us,  0.3 sy,  0.0 ni, 98.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  : 17.7 us,  1.3 sy,  0.0 ni, 81.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :  15874.0 total,   3964.8 free,   3177.2 used,   8732.0 buff/cache
MiB Swap:   8096.0 total,   8086.7 free,      9.3 used.  11455.8 avail Mem

    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 15.4: **Using top**

© Copyright the Linux Foundation 2023. All rights reserved.
THE LINUX FOUNDATION Training & Certification

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.

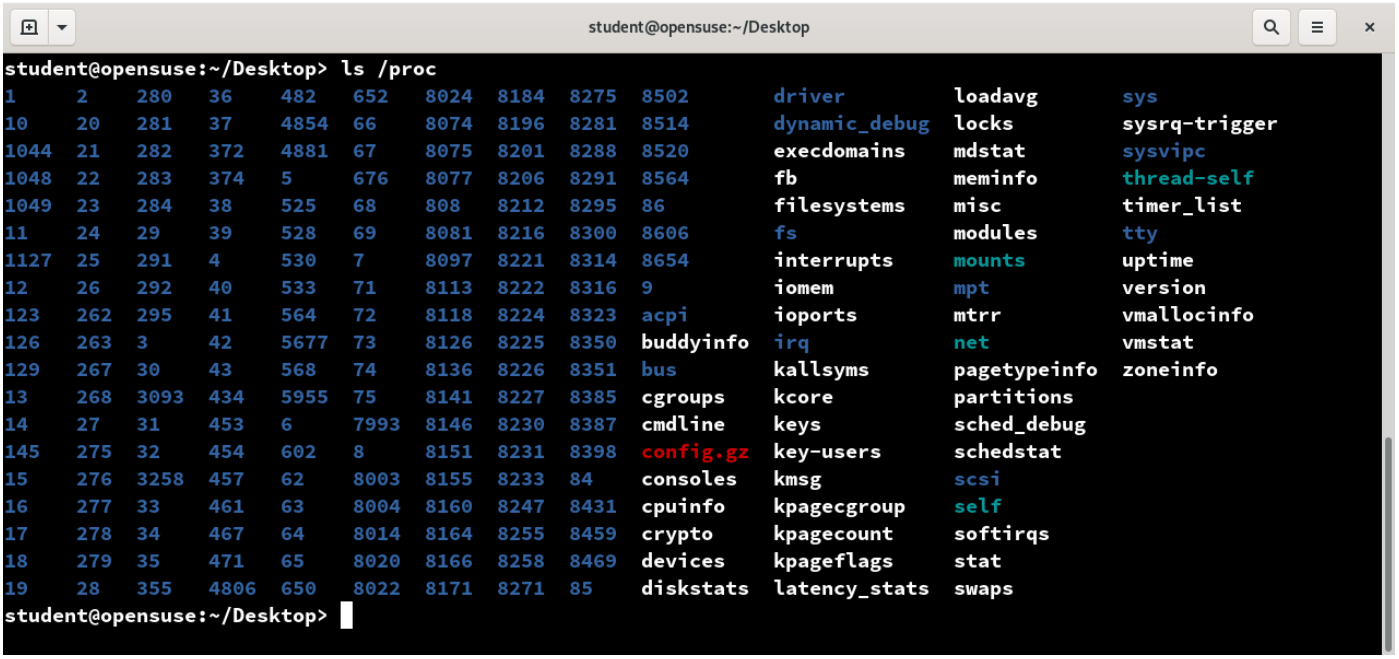


Figure 15.5: /**proc Contents**

© Copyright the Linux Foundation 2023. All rights reserved.

15.6 Labs

Video Demonstration Resources

top_demo.mp4
ps_demo.mp4

Exercise 15.1: 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 15.1

1.
```
$ ps -ef
$ ps aux
```

2.
```
$ ps -o pid,pri,ni,cmd
```

```
  PID PRI  NI CMD
 2389  19   0 bash
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
```

```
 2389  19   0 bash
22115  19   0 bash
22171   9  10 bash
22227   9  10 ps -o pid,pri,ni,cmd
```

4.
```
$ renice 15 -p 22171
$ ps -o pid,pri,ni,cmd
```

```
  PID PRI  NI CMD
 2389  19   0 bash
22115  19   0 bash
22171   4  15 bash
22246   4  15 ps -o pid,pri,ni,cmd
```


© Copyright the Linux Foundation 2023. All rights reserved.

5. `$ top`

Exercise 15.2: 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?
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 15.2

1. `$ dd if=/dev/urandom of=/dev/null &`
2. `$ ps -C dd -o pid,cmd,stat`

```
  25899 dd if=/dev/urandom of=/dev/ R
```

   Should be `S` or `R`.

3.
```
$ fg
$ ^Z
$ ps -C dd -o pid,cmd,stat
```

```
    PID CMD                         STAT
  25899 dd if=/dev/urandom of=/dev/ T
```

   State should be `T`.

4. Type the **jobs** command. What does it tell you?

```
$ jobs
```

```
  [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
```


© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 16

Memory Monitoring, Usage and Configuring Swap

16.1 Memory Monitoring and Tuning 250
16.2 /proc/sys/vm 252
16.3 vmstat 253
16.4 Swap 254
16.5 Out of Memory Killer (OOM) 255
16.6 Labs 257

Learning Objectives

By the end of this session, you should be able to:

- List the primary (inter-related) considerations and tasks involved in memory tuning.
- Use entries in `/proc/sys/vm`, and decipher `/proc/meminfo`.
- Use **vmstat** to display information about memory, paging, I/O, processor activity, and processes' memory consumption.
- Understand how **swap** areas are used, and how to create, monitor and regulate them.
- Understand how the **OOM-killer** decides when to take action and selects which processes should be exterminated to open up some memory

16.1 Memory Monitoring and Tuning

Memory Monitoring

Important tools for monitoring and tuning memory in **Linux**

Table 16.1: **Memory Monitoring Utilities**

Utility	Purpose	Package
free	Brief summary of memory usage	procps
vmstat	Detailed virtual memory statistics and block I/O, dynamically updated	procps
pmap	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.

© Copyright the Linux Foundation 2023. All rights reserved.

/proc/meminfo

```
$ cat /proc/meminfo
```

```
MemTotal:       16254132 kB     Writeback:             0 kB     VmallocChunk:          0 kB
MemFree:          162652 kB     AnonPages:       2780004 kB     Percpu:             8352 kB
MemAvailable:   12234416 kB     Mapped:           679632 kB     HardwareCorrupted:     0 kB
Buffers:          847972 kB     Shmem:            523880 kB     AnonHugePages:    958464 kB
Cached:         10959104 kB     KReclaimable:    1124064 kB     ShmemHugePages:        0 kB
SwapCached:        13276 kB     Slab:            1233300 kB     ShmemPmdMapped:        0 kB
Active:          3252848 kB     SReclaimable:    1124064 kB     FileHugePages:         0 kB
Inactive:       11343484 kB     SUnreclaim:       109236 kB     FilePmdMapped:         0 kB
Active(anon):     393416 kB     KernelStack:       16512 kB     HugePages_Total:       0
Inactive(anon):  2919760 kB     PageTables:        79092 kB     HugePages_Free:        0
Active(file):    2859432 kB     NFS_Unstable:          0 kB     HugePages_Rsvd:        0
Inactive(file):  8423724 kB     Bounce:                0 kB     HugePages_Surp:        0
Unevictable:       75032 kB     WritebackTmp:          0 kB     Hugepagesize:       2048 kB
Mlocked:              32 kB     CommitLimit:    16417364 kB     Hugetlb:               0 kB
SwapTotal:       8290300 kB     Committed_AS:   10351404 kB     DirectMap4k:      497740 kB
SwapFree:        7911980 kB     VmallocTotal:   34359738367 kB  DirectMap2M:    16152576 kB
Dirty:                60 kB     VmallocUsed:       28256 kB     DirectMap1G:     1048576 kB
```

The pseudo file `/proc/meminfo` contains a wealth of information about how memory is being used.

© Copyright the Linux Foundation 2023. All rights reserved.
THE LINUX FOUNDATION | Training & Certification

16.2 /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.

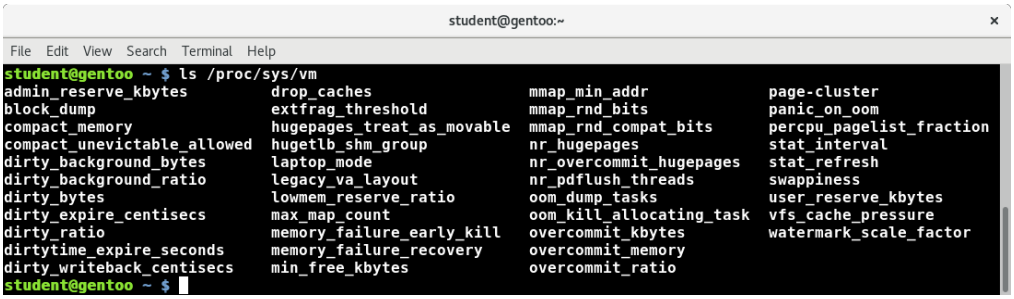


Figure 16.1: /**proc/sys/vm**

© Copyright the Linux Foundation 2023. All rights reserved.

16.3 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.

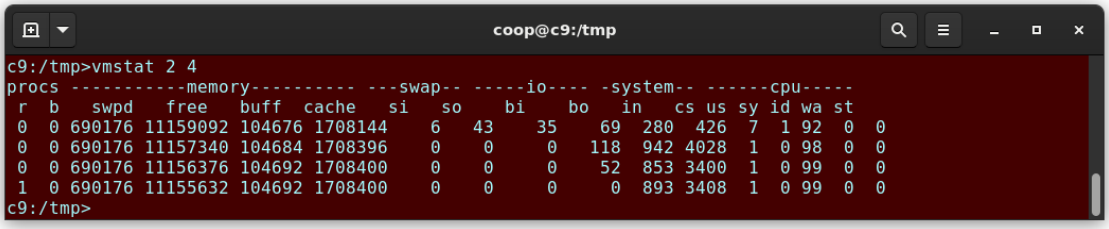


Figure 16.2: **vmstat**

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.

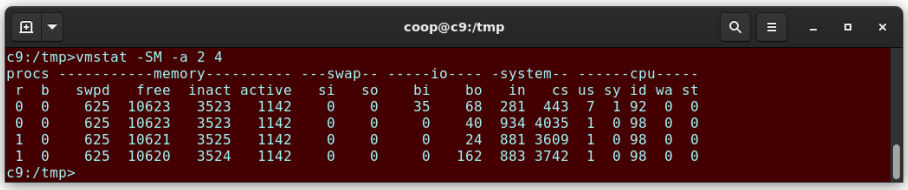


Figure 16.3: **Using vmstat**

If you just want to get some quick statistics on only one partition, use the `-p` option

Figure 16.4: **Using vmstat on One Disk**

© Copyright the Linux Foundation 2023. All rights reserved.

16.4 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 **overcommission** 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** (**C**opy **O**n **W**rite) 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.

© Copyright the Linux Foundation 2023. All rights reserved.

16.5 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.

© Copyright the Linux Foundation 2023. All rights reserved.

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 overcommission by setting the value of `/proc/sys/vm/overcommit_memory`:

- 0: (default) Permit overcommission, 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 overcommission. 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.

© Copyright the Linux Foundation 2023. All rights reserved.

16.6 Labs

Video Demonstration Resources

using_swap_and_oom_demo.mp4

Exercise 16.1: Managing Swap Space

Examine your current swap space by doing:

```
$ cat /proc/swaps
```

```
Filename                                Type            Size    Used    Priority
/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
```

```
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 1.30576 s, 822 MB/s
```

```
$ mkswap swpfile
```

```
Setting up swapspace version 1, size = 1048572 KiB
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
```

```
swapon: /tmp/swpfile: insecure permissions 0664, 0600 suggested.
swapon: /tmp/swpfile: insecure file owner 500, 0 (root) suggested.
```

On Red Hat, 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
```

```
Filename                                Type            Size    Used    Priority
/dev/sda11                              partition       4193776 0       -1
/tmp/swpfile                            file            1048572 0       -2
```


© Copyright the Linux Foundation 2023. All rights reserved.

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:

```
$ sudo swapoff swpfile
$ sudo rm swpfile
```

Exercise 16.2: Invoking the OOM Killer

- When the **Linux** kernel gets under extreme memory pressure it invokes the dreaded **OOM** (**O**ut **O**f **M**emory) **Killer**. This tries to select the "best" process to kill to help the system recover gracefully.
- We are going to force the system to run short on memory and watch what happens. The first thing to do is to open up a terminal window, and in it type:

  ```
  $ sudo tail -f /var/log/messages
  ```

  in order to watch kernel messages as they appear.

  - An even better way to look is furnished by:

    ```
    $ dmesg -w
    ```

    as it does not show non-kernel messages.

- This exercise will be easier to perform if we turn off all swap first with the command:

  ```
  $ sudo /sbin/swapoff -a
  ```

  Make sure you turn it back on later with

  ```
  $ sudo /sbin/swapon -a
  ```

- Now we are going to put the system under increasing memory pressure. You are welcome to find your own way of doing it but we also supply a program for consuming the memory:

lab_wastemem.c

```
1  /* simple program to defragment memory, J. Cooperstein 2/04
2   */
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <unistd.h>
7  #include <string.h>
8  #include <sys/sysinfo.h>
9  #include <signal.h>
10
11 #define MB (1024*1024)
12 #define BS 16                          /* will allocate BS*MB at each step */
13 #define CHUNK (MB*BS)
14 #define QUIT_TIME 20
15 void quit_on_timeout(int sig)
16 {
17         printf("\n\nTime expired, quitting\n");
18         exit(EXIT_SUCCESS);
19 }
20
21 int main(int argc, char **argv)
22 {
```


© Copyright the Linux Foundation 2023. All rights reserved.

```c
        struct sysinfo si;
        int j, m;
        char *c;

        /* get total memory on the system */
        sysinfo(&si);
        m = si.totalram / MB;
        printf("Total System Memory in MB = %d MB\n", m);
        m = (9 * m) / 10;        /* drop 10 percent */
        printf("Using somewhat less: %d MB\n", m);

        if (argc == 2) {
                m = atoi(argv[1]);
                printf("Choosing instead mem = %d MB\n", m);
        }

        signal(SIGALRM, quit_on_timeout);
        printf("Will quite in QUIT_TIME seconds if no normal termination\n");
        alarm(QUIT_TIME);

        for (j = 0; j <= m; j += BS) {
                /* yes we know this is a memory leak, no free,
                 * that's the idea!
                 */
                c = malloc(CHUNK);
                /* just fill the block with j over and over */
                memset(c, j, CHUNK);
                printf("%8d", j);
                fflush(stdout);
        }
        printf("\n\n    Sleeping for 5 seconds\n");
        sleep(5);
        printf("\n\n    Quitting and releasing memory\n");
        exit(EXIT_SUCCESS);
}
```

It takes as an argument how many MB to consume. Keep running it, gradually increasing the amount of memory requested until your system runs out of memory.

Please Note

You should be able to compile the program and run it by just doing:

```
$ gcc -o lab_wastemem lab_wastemem.c
$ ./lab_wastemem 4096
```

which would waste 4 GB. It would be a good idea to run **gnome-system-monitor** or another memory monitoring program while it is running (although the display may freeze for a while!)

- You should see the **OOM** (Out of Memory) killer swoop in and try to kill processes in a struggle to stay alive. Who gets clobbered first?

Solution 16.2

lab_wastemem.c
lab_waste.sh

© Copyright the Linux Foundation 2023. All rights reserved.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 17

I/O Monitoring and Tuning

17.1 I/O Monitoring 262
17.2 iostat 263
17.3 iotop 264
17.4 Labs 265

Learning Objectives

By the end of this session, you should be able to:

- Understand the importance of monitoring I/O activity and when it constitutes system performance bottlenecks.
- Use **iostat** to monitor system I/O device activity.
- Use **iotop** to display a constantly updated table of current I/O usage.

17.1 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
- Some important tools:
  - **iostat**
  - **iotop**

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.

© Copyright the Linux Foundation 2023. All rights reserved.

17.2 iostat

iostat

- Basic utility for monitoring all I/O activity
- Very flexible report options

```
$ iostat [OPTIONS]  [devices] [interval] [count]
```

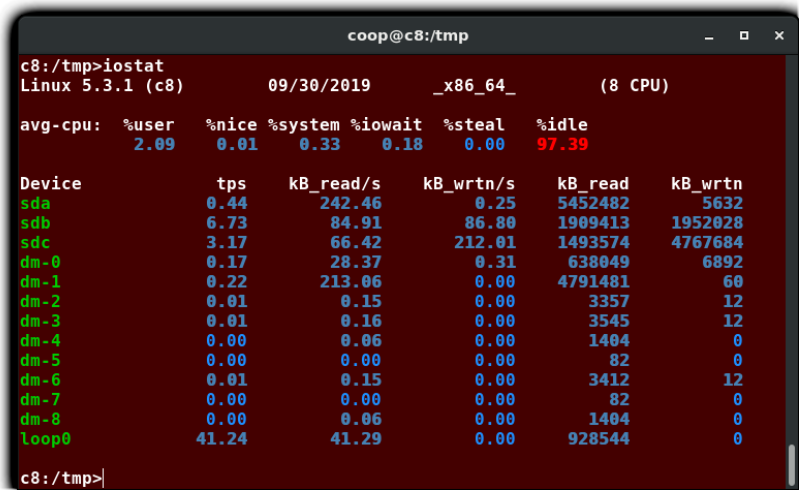


Figure 17.1: **iostat**

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:

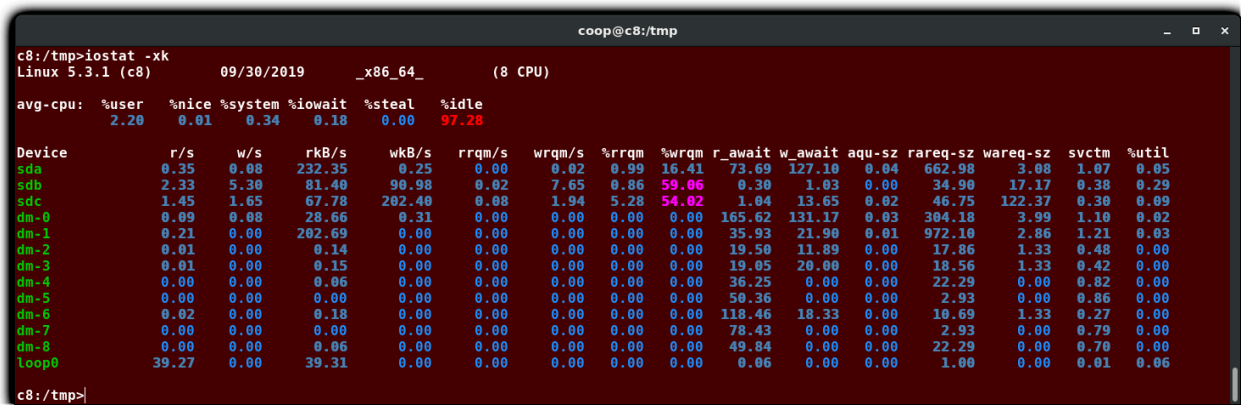


Figure 17.2: **Using iostat**

© Copyright the Linux Foundation 2023. All rights reserved.

17.3 iotop

iotop

- **iotop** displays current I/O activity
- Updates periodically like **top**
- Using the `-o` option can be useful to avoid clutter.

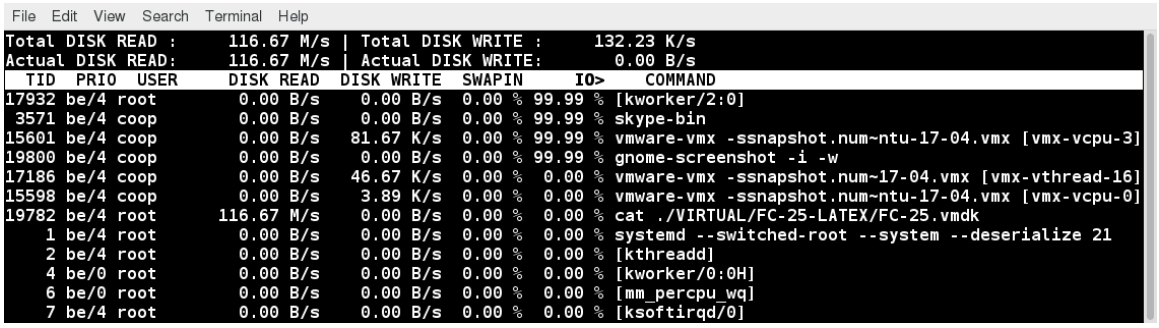


Figure 17.3: **Using iotop**

Another very useful utility is **iotop**, which must be run as root. It displays a table of current I/O usage and updates periodically. In the `PRIO` column, **be** stands for **best effort** and **rt** stands for **real time**.

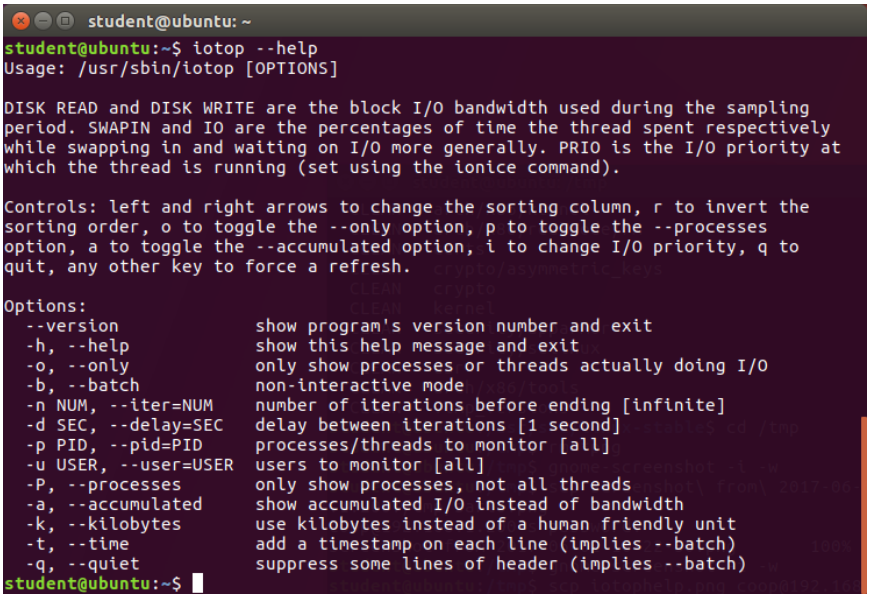


Figure 17.4: **iotop options**

© Copyright the Linux Foundation 2023. All rights reserved.

17.4 Labs

Exercise 17.1: 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 (**c**omma **s**eparated **v**alue). Companion programs, **bon_csv2html** and **bon_csv2txt**, can be used to 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
```

```
bonnie++: invalid option -- '-'
usage:
bonnie++ [-d scratch-dir] [-c concurrency] [-s size(MiB)[:chunk-size(b)]]
      [-n number-to-stat[:max-size[:min-size][:num-directories[:chunk-size]]]]
      [-m machine-name] [-r ram-size-in-MiB]
      [-x number-of-tests] [-u uid-to-use:gid-to-use] [-g gid-to-use]
      [-q] [-f] [-b] [-p processes | -y] [-z seed | -Z random-file]
      [-D]

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 Red Hat

For **RHEL/CentOS 9**, there is currently no package available but the earlier version works, and can be obtained from the **EPEL 8** repository: https://dl.fedoraproject.org/pub/epel/8/Everything/x86_64/Packages/b/bonnie++-1.98-1.el8.x86_64.rpm and install with:

```
$ sudo dnf localinstall bonnie++-*rpm
```

On an **RHEL/CentOS 8** system:

© Copyright the Linux Foundation 2023. All rights reserved.

```
c8:/tmp>sudo bonnie++ -n 0 -u 0 -r 100 -f -b -d /mnt
```

```
Using uid:0, gid:0.
Writing intelligently...done
Rewriting...done
Reading intelligently...done
start 'em...done...done...done...done...done...
Version  1.98       ------Sequential Output------ --Sequential Input- --Random-
                    -Per Chr- --Block-- -Rewrite- -Per Chr- --Block-- --Seeks--
Name:Size etc        /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP
c8             300M            271m  20  288m  15           +++++ +++  3916  22
Latency                          30us      20us                12us   18075us

1.98,1.98,c8,1,1616174245,300M,,8192,5,,,277062,20,294543,15,,,+++++,+++,3916,22,,,,,,,,,,,,,,,,,,,\
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
```

```
sing uid:0, gid:0.
Writing intelligently...done
Rewriting...done
Reading intelligently...done
start 'em...done...done...done...done...done...
Version  1.98       ------Sequential Output------ --Sequential Input- --Random-
                    -Per Chr- --Block-- -Rewrite- -Per Chr- --Block-- --Seeks--
Name:Size etc        /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP
ubuntu         300M            287m  88 +++++ +++           +++++ +++ +++++ +++
Latency                        3805us    4650us              2136us    1852us

1.98,1.98,ubuntu,1,1616156268,300M,,8192,5,,,293955,88,+++++,+++,,,+++++,+++,+++++,+++,,,,,,,,,,,,,,\
,,,,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 17.2: 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:

© Copyright the Linux Foundation 2023. All rights reserved.

```
$ 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
```

```
....
/usr/bin/ld: cannot find -lc
```

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 Red Hat, 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.

© Copyright the Linux Foundation 2023. All rights reserved.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 18

Containers Overview

18.1 Containers 270
18.2 Containers vs Virtual Machines 271
18.3 Docker 272
18.4 Cowsay Example 275
18.5 Reproducible Builds 279
18.6 Labs 283

Learning Objectives

By the end of this session, you should be able to:

- Understand the basic parameters and methods that define containers.
- Know what it means to virtualize an application.
- Understand the difference between containers and virtual machines.
- Learn to use **Docker** with a detailed example of a adding a small utility program to the system.
- Analyze the problems with version specifications as an intro to **Reproducible Builds**.

18.1 Containers

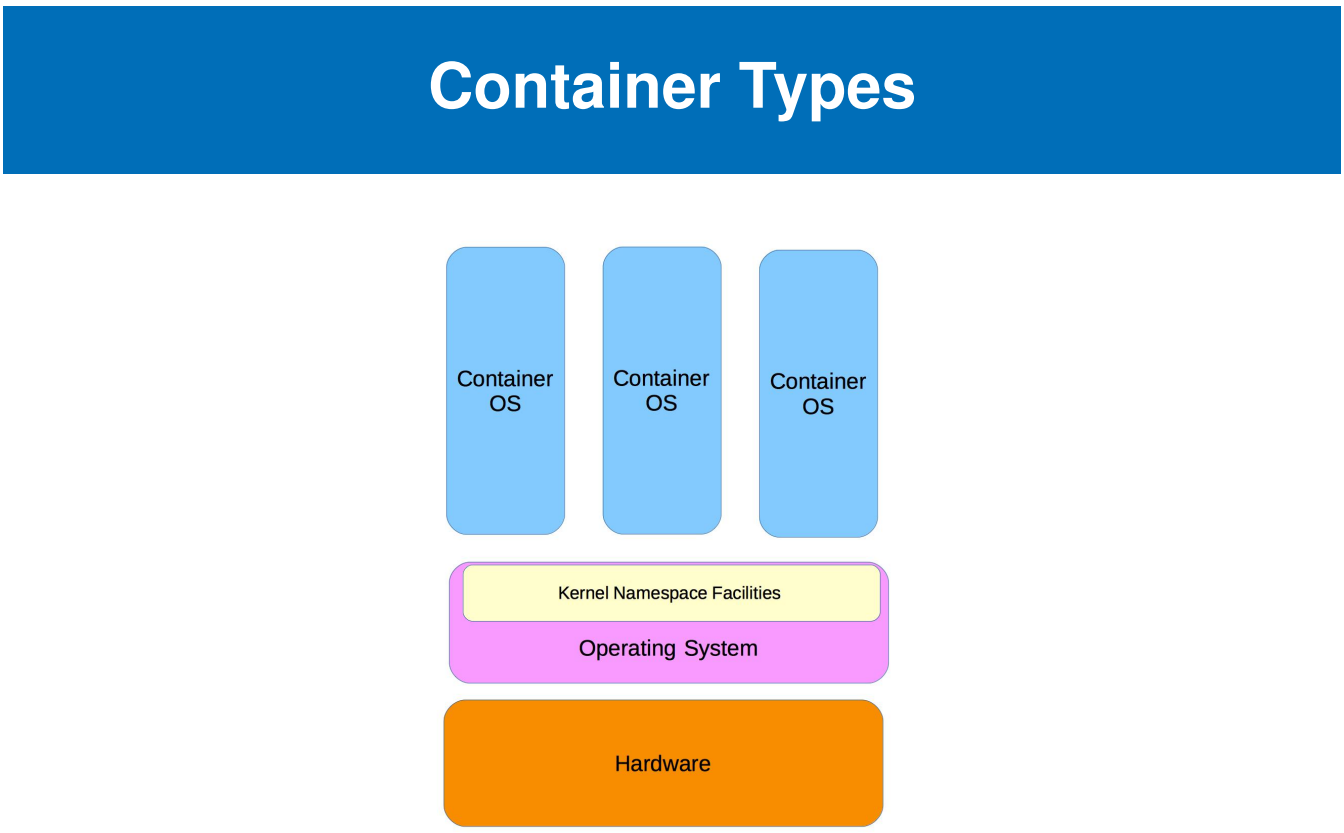


Figure 18.1: **Linux Containers vs OCI Containers**

Using **Linux** kernel features including **namespaces** and **cgroups**, it is possible to provide a lightweight instance of an OS or application that shares the host **kernel**. Initially, **Linux Containers** or **LXC** was created to allow an entire virtual OS with an init, an entire set of system tools, and also an application to be virtualized as a container.

Next came **Application Containers**, starting with **Docker** and then generically referred as OCI containers, these containers typically do not include an entire OS (although but they can). enabling smaller images, often with just one application or function.

Today there are many OCI-compatible technologies to run such application containers including: *podman*, *runc*, *firecracker*, *apptainer*, and *rkt*.

We are going to focus on **Docker** and the **docker compose** functionality in this chapter, although most of the techniques translate directly to other OCI runtimes. Generally images in **container registries** that specify **Docker** can be used in any other runtime with minor modifications.

© Copyright the Linux Foundation 2023. All rights reserved.

18.2 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 has had its day in the sun and been 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.

© Copyright the Linux Foundation 2023. All rights reserved.

18.3 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.

© Copyright the Linux Foundation 2023. All rights reserved.

Find a Docker image

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)`.

© Copyright the Linux Foundation 2023. All rights reserved.

Extend a Docker image

Often you want to modify an existing container image, either to provide different configuration values or to add more functionality. There are a few ways to accomplish this:

- Creating a new image based on another image with the `FROM` command in the `Dockerfile`
- Using **docker-compose**
- Finding the source `Dockerfile` for an image and modifying it as the base of your projeect

In most instances, you will inherit some other image with the `FROM` command, even if it is just the base operating system image.

For security, or internal compliance reasons this could include a local source or more base operating system steps.

© Copyright the Linux Foundation 2023. All rights reserved.

18.4 Cowsay Example

A Cow Says

Here's a very basic example using the *cowsay* command, which simply prints some text in a silly way.

cowsay-minimal.Dockerfile

```
# moo
FROM alpine:edge
RUN apk add --no-cache -X http://dl-cdn.alpinelinux.org/alpine/edge/testing cowsay
ENTRYPOINT ["cowsay"]
```

```
$ docker run --rm cowsay-minimal 'and.. Action!'
```

```
 ---------------
< and.. Action! >
 ---------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Our example uses **Alpine Linux**, which is often used as a base for containers because of its small size.

Alpine Linux differs from the mainline distributions we've covered so far in several ways, including:

- designed to run from read-only media by default
- uses **musl** instead of **glibc** as its **Standard C Library**
- uses a completely different package manager

Want More?

We aren't going to cover this distribution in detail, but you can get more information from:

1. **Alpine Linux** – https://alpinelinux.org/about/
2. The **musl** C library – https://musl.libc.org/
3. The **apk** package manager – https://wiki.alpinelinux.org/wiki/Alpine_Package_Keeper

© Copyright the Linux Foundation 2023. All rights reserved.
THE LINUX FOUNDATION | Training & Certification

How Cow

Let's break this down:

```
# moo
FROM alpine:edge
```

For the **base** of our container, use the rolling release of the OS.

```
RUN apk add --no-cache -X http://dl-cdn.alpinelinux.org/alpine/edge/testing cowsay
```

Use the system package command to install `cowsay`; from a non-default package repository; and don't save any packaging information for next time.

```
ENTRYPOINT ["cowsay"]
```

Set the default command that **docker run** will use as the `cowsay` command.

In the `RUN` step above, we included a very complex command. Could we not have done the same thing with multiple commands for clarity?

There is a heavy emphasis on the size and makeup of a docker image and most people agree that:

- smaller size is better
- fewer **layers** are better

Each `RUN` line in the `Dockerfile` represents a command that is executed **atomically**. Each of these actions is saved in a snapshot called a **layer**, and all of those layers have to be downloaded to create the final image.

There are a few techniques to accomplish fewer layers:

1. doing many actions in one line, sometimes using the `&&` shell operator
2. build systems that handle this detail automatically
3. **multi-stage builds**

© Copyright the Linux Foundation 2023. All rights reserved.

Build Cow

Once we've created our **Dockerfile**, we need to build it:

```
$ docker build  -f cowsay-minimal.Dockerfile -t  cowsay-minimal  .
```

```
Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM alpine:edge
 ---> 855f10c27d71
Step 2/3 : RUN apk add --no-cache -X http://dl-cdn.alpinelinux.org/alpine/edge/testing
↪  cowsay
 ---> Using cache
 ---> 8d2935a86a26
Step 3/3 : ENTRYPOINT ["cowsay"]
 ---> Using cache
 ---> 90a4fc4e505d
Successfully built 90a4fc4e505d
Successfully tagged cowsay-minimal:latest
```

We specified some arguments to change the default behavior of **docker build**:

-f, --file path/file name of `Dockerfile` to use

-t, --tag the name of the tag to be created on a successful build

.

the last required argument is the working directory. In this case, the current one.

© Copyright the Linux Foundation 2023. All rights reserved.

Run Cow

Once we have our docker **image**, we can use it to interact with information from our system.

```
docker run --rm cowsay-minimal "$(cat /etc/debian_version)"
```

```
 ------
< 11.7 >
 ------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

The breakdown of our **docker run** command:

--rm remove the **container** created from the **image** so we start fresh each time

`cowsay-minimal`

the image name, or in our case **tag** to specify which specific image to run

`"$(cat /etc/debian_version)"`

the rest of the arguments passed to the docker `ENTRYPOINT` program. In this case it was a single argument as the result a subshell run on our host.

Very Important

Dynamic arguments are going to be processed before being passed to docker.

Please Note

When you are running a command that will execute another command, it is often a good practice to specify the end of the argument list to the first program with the **--** argument.
So our run command could also have been written

```
docker run --rm cowsay-minimal -- "$(cat /etc/debian_version)"
```


© Copyright the Linux Foundation 2023. All rights reserved.

18.5 Reproducible Builds

Wrong Cow

One of the arguments for using containers is the ability to create the same environment on multiple machines – getting rid of the *"works on my machine"* problem. However, just using docker or another container system doesn't fully provide this guarantee. Let's go back to our example:

cowsay-minimal.Dockerfile

```
# moo
FROM alpine:edge
#    ^^^^^^^^^^^
```

Ignoring the rest for the moment, we previously noted that `edge` was the name of the **rolling release**, similar to **Debian** `Testing` or **CentOS** `Stream`. These types of distributions frequently get updates, sometimes nightly. If we build the same **docker image** tomorrow, how different might it be?

Even if we were to specify a different version above instead of `edge`, it is still possible that a security update or some other change might be published. If we are using **docker** as a base for developing software, an unexpected change in that **docker image** might appear to be a change that we made to our project's code or configuration.

© Copyright the Linux Foundation 2023. All rights reserved.

Be More Specific

Our `FROM` line can specify more than a version, it can specify a **digest** with a `sha256` hash:

cowsay-digest.Dockerfile

```
#moo
FROM alpine:edge@sha256:2d01a16bab53a8405876cec4c27235d47455a7b72b75334c614f2fb0968b3f90
RUN apk add --no-cache -X http://dl-cdn.alpinelinux.org/alpine/edge/testing cowsay
ENTRYPOINT ["cowsay"]
```

We've made a copy of our `Dockerfile`, specified the digest that went with our current image, and fired off our build step again:

```
$ docker build -f cowsay-digest.Dockerfile -t cowsay-digest .
```

```
### very similar output follows ###
```

It's usually fastest to find the digest hash for a new image with a web browser instead of the docker command line tools.

We are specifying both the docker **tag** or **label** here, but some **OCI** container programs work with *just* or *only* the `sha256` hash.

© Copyright the Linux Foundation 2023. All rights reserved.

Compare Images

```
$ docker images
```

```
REPOSITORY                  TAG                 IMAGE ID       CREATED          SIZE
cowsay-digest               latest              90a4fc4e505d   13 minutes ago   43.2MB
cowsay-minimal              latest              90a4fc4e505d   13 minutes ago   43.2MB
```

We can see that we created the same image as far as docker is concerned by using the extra hash information. Now if a new version is published to `alpine:edge`, we don't have to worry about accidentally upgrading.

Very Important

If you do declare (*pin*) a specific version it means you won't accidentally upgrade, but it is important to have some sort of upgrade procedure to increase (*bump*) so you receive important security patches.

Several InfoSec companies have issued reports showing that around **half** of all published, public containers have security vulnerabilities, some very serious.

© Copyright the Linux Foundation 2023. All rights reserved.

But What About

It may have occurred to you that the base image of the software wasn't the only thing that could change in our example:

cowsay-digest.Dockerfile

```
RUN apk add --no-cache -X http://dl-cdn.alpinelinux.org/alpine/edge/testing cowsay
```

We are installing a program that isn't part of the base system, could it not change as well?

Yes. And in fact if you want to create a **bit-by-bit** identical version of an image, there are considerations such as temporary files, and even the timestamps on the files in the filesystem.

It's a difficult problem, but the benefits are many, and there has already been significant work done in this problem space.

Want More?

Reproducible Builds

See https://reproducible-builds.org/ for much more information.

One of the methods of providing the same package version even after a rebuild from the same source where the timestamps would have changed, is to make use of a "snapshot server" that makes a copy of all of the packages for a distribution at a specific moment in time.

Check out this blog post (https://www.qubes-os.org/news/2021/10/08/reproducible-builds-for-debian-a-big-step-forward/) for a discussion on how this was accomplished for **Debian**.

© Copyright the Linux Foundation 2023. All rights reserved.

18.6 Labs

Exercise 18.1: Make a small utility available with docker

Taking our `cowsay` example, build a similar small docker container for the **httpie** utility (https://github.com/httpie/httpie).

Solution 18.1

Alpine Linux solution:

1. **Verify the package name**
   If you just guessed it was "httpie", that is going to work fine. Also, you can use the package search at: https://pkgs.alpinelinux.org/package/edge/community/x86_64/httpie or check the information in the upstream `README`.
2. **Create a Dockerfile**
   You can use either the `minimal` or `digest` version for this, but in this instance, you probably would use the `minimal` version in practice because the latest version on each build would be desired.
   Additionally, you don't need the extra repo argument for `testing` but it won't hurt anything having it there.

httpie.Dockerfile

```
FROM alpine:edge
RUN apk add --no-cache httpie
CMD ["httpie"]
```

3. **Build the image**

```
$ docker build  -f httpie.Dockerfile -t  httpie:latest .
```

```
Sending build context to Docker daemon  30.72kB
Step 1/3 : FROM alpine:edge
 ---> 855f10c27d71
Step 2/3 : RUN apk add --no-cache httpie
 ---> Running in 2f6b1436e5e3
fetch https://dl-cdn.alpinelinux.org/alpine/edge/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/edge/community/x86_64/APKINDEX.tar.gz
(1/57) Installing libbz2 (1.0.8-r6)
(2/57) Installing libexpat (2.5.0-r2)
(3/57) Installing libffi (3.4.4-r3)
(4/57) Installing gdbm (1.23-r1)
(5/57) Installing xz-libs (5.4.3-r1)
(6/57) Installing libgcc (13.1.1_git20230617-r0)
(7/57) Installing libstdc++ (13.1.1_git20230617-r0)
(8/57) Installing mpdecimal (2.5.1-r2)
(9/57) Installing ncurses-terminfo-base (6.4_p20230625-r0)
(10/57) Installing libncursesw (6.4_p20230625-r0)
(11/57) Installing libpanelw (6.4_p20230625-r0)
(12/57) Installing readline (8.2.1-r2)
(13/57) Installing sqlite-libs (3.42.0-r2)
(14/57) Installing python3 (3.11.4-r0)
(15/57) Installing python3-pycache-pyc0 (3.11.4-r0)
(16/57) Installing pyc (0.1-r0)
(17/57) Installing py3-multidict (6.0.4-r2)
(18/57) Installing py3-multidict-pyc (6.0.4-r2)
(19/57) Installing py3-parsing (3.1.0-r0)
(20/57) Installing py3-parsing-pyc (3.1.0-r0)
(21/57) Installing py3-packaging (23.1-r1)
(22/57) Installing py3-packaging-pyc (23.1-r1)
(23/57) Installing py3-setuptools (68.0.0-r0)
(24/57) Installing py3-setuptools-pyc (68.0.0-r0)
```


© Copyright the Linux Foundation 2023. All rights reserved.

```
(25/57) Installing py3-pip (23.1.2-r0)
(26/57) Installing py3-pip-pyc (23.1.2-r0)
(27/57) Installing py3-pygments (2.15.1-r0)
(28/57) Installing py3-pygments-pyc (2.15.1-r0)
(29/57) Installing py3-pysocks (1.7.1-r5)
(30/57) Installing py3-pysocks-pyc (1.7.1-r5)
(31/57) Installing py3-certifi (2023.5.7-r0)
(32/57) Installing py3-certifi-pyc (2023.5.7-r0)
(33/57) Installing py3-charset-normalizer (3.1.0-r1)
(34/57) Installing py3-charset-normalizer-pyc (3.1.0-r1)
(35/57) Installing py3-idna (3.4-r4)
(36/57) Installing py3-idna-pyc (3.4-r4)
(37/57) Installing py3-urllib3 (1.26.15-r2)
(38/57) Installing py3-urllib3-pyc (1.26.15-r2)
(39/57) Installing py3-requests (2.31.0-r0)
(40/57) Installing py3-requests-pyc (2.31.0-r0)
(41/57) Installing py3-requests-toolbelt (1.0.0-r0)
(42/57) Installing py3-requests-toolbelt-pyc (1.0.0-r0)
(43/57) Installing py3-attrs (23.1.0-r1)
(44/57) Installing py3-attrs-pyc (23.1.0-r1)
(45/57) Installing py3-mdurl (0.1.2-r2)
(46/57) Installing py3-mdurl-pyc (0.1.2-r2)
(47/57) Installing py3-markdown-it-py (3.0.0-r0)
(48/57) Installing py3-markdown-it-py-pyc (3.0.0-r0)
(49/57) Installing py3-rich (13.4.2-r0)
(50/57) Installing py3-rich-pyc (13.4.2-r0)
(51/57) Installing py3-wheel (0.40.0-r1)
(52/57) Installing py3-wheel-pyc (0.40.0-r1)
(53/57) Installing httpie-pyc (3.2.2-r0)
(54/57) Installing py3-defusedxml-pyc (0.7.1-r4)
(55/57) Installing python3-pyc (3.11.4-r0)
(56/57) Installing py3-defusedxml (0.7.1-r4)
(57/57) Installing httpie (3.2.2-r0)
Executing busybox-1.36.0-r5.trigger
OK: 95 MiB in 72 packages
Removing intermediate container 2f6b1436e5e3
 ---> 227f4a0c96ab
Step 3/3 : ENTRYPOINT ["httpie"]
 ---> Running in f19a66bab6cb
Removing intermediate container f19a66bab6cb
 ---> c0b613745c84
Successfully built c0b613745c84
Successfully tagged httpie:latest
```

4. **Run the image**

```
$ docker run --rm httpie
```

```
usage: httpie [-h] [--debug] [--traceback] [--version] {cli,plugins} ...
httpie: error: Please specify one of these: 'cli', 'plugins'

This command is only for managing HTTPie plugins.
To send a request, please use the http/https commands:

  $ http POST pie.dev/post hello=world

  $ https POST pie.dev/post hello=world
```

And then try the suggested command:

© Copyright the Linux Foundation 2023. All rights reserved.

```
$ docker run --rm httpie https
```

```
usage:
    http [METHOD] URL [REQUEST_ITEM ...]

error:
    the following arguments are required: URL

for more information:
    run 'http --help' or visit https://httpie.io/docs/cli
```

5. *(Optional)* **Add it to your aliases**

```
$ alias https='docker run --rm httpie https -- '
```

```
$ https httpie.io/hello
{"ahoy":["Hello, World!  Thank you for trying out HTTPie ","We hope this will become a
↪  friendship."],"
↪  links":{"homepage":"https://httpie.io","twitter":"https://twitter.com/httpie",
    "discord":"https://httpie.io/discord","github":"https://github.com/httpie"}}
```


© Copyright the Linux Foundation 2023. All rights reserved.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 19

Linux Filesystems and the VFS

19.1 Filesystem Basics 288
19.2 Filesystem Concepts 289
19.3 Virtual Filesystem (VFS) 291
19.4 Available Filesystems 292
19.5 Journalling Filesystems 294
19.6 Special Filesystems 295
19.7 Labs 296

Learning Objectives

By the end of this session, you should be able to:

- Explain the basic filesystem organization.
- Understand the role of the VFS (Virtual File System).
- Know what filesystems are available in **Linux** and which ones can be used on your actual system.
- Grasp why journalling filesystems represent significant advances.
- Discuss the use of special filesystems in **Linux**.

19.1 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.

© Copyright the Linux Foundation 2023. All rights reserved.

19.2 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.

Note that there are three timestamps:

1. **Access Time**: The last time the file was accessed for any purpose.
2. **Modification Time**: The last time file's **contents** were modified.
3. **Change Time**: The last time the file's inode was changed, by a change in permissions, ownership, filename, hard links etc.

© Copyright the Linux Foundation 2023. All rights reserved.

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

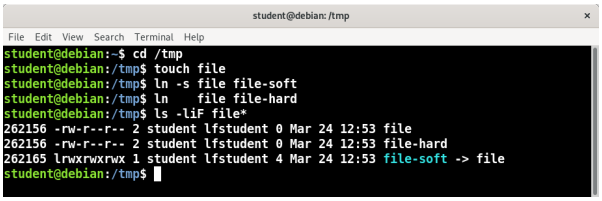


Figure 19.1: **Hard and Soft Links**

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 by 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**.

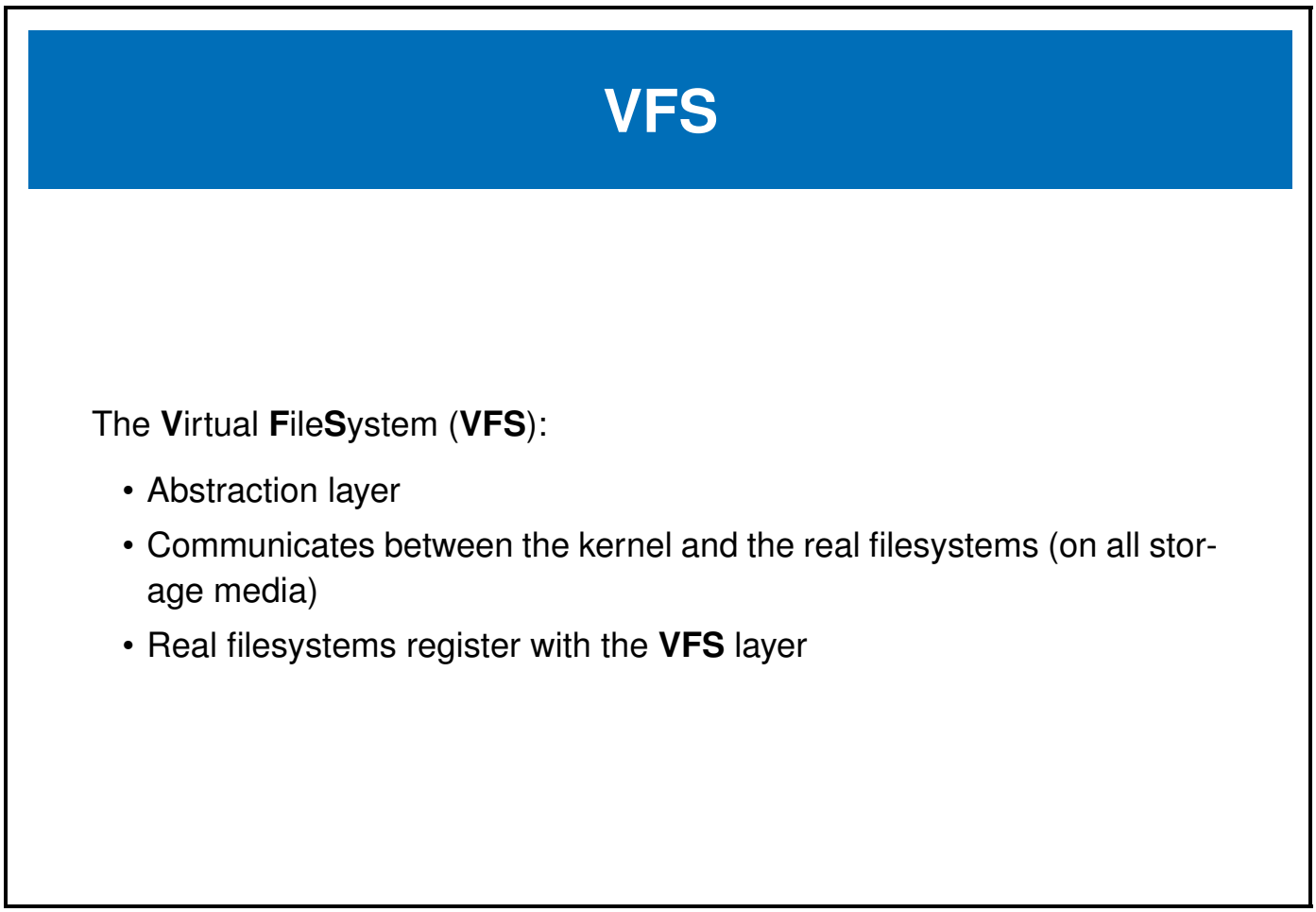
Because it is possible (and quite common) for two or more directory entries to point to the same inode (hard links), a file can be known by multiple names, each of which has its own place in the directory structure. However, it can have only one inode no matter which name is being used.

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.

Normally when you modify a file it does not break the hard links that reference the same inode. However, there are (badly written) applications that can copy a file and change it and then replace it, or delete a file and replace it, and in the process create a new file that is not linked any more. So keep your eye out for this behavior if it is not intended.

© Copyright the Linux Foundation 2023. All rights reserved.

19.3 Virtual Filesystem (VFS)



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.

© Copyright the Linux Foundation 2023. All rights reserved.

19.4 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.
- Native (kernel) filesystems tend to have full read/write access, while a those from other operating systems or specialized ones may have only read access or experimental write access.
- Some **key-value** filesystems (like S3-compatible) ones might not even allow every file operation and are often available via FUSE in userspace.
- Commonly used filesystems include **ext4, xfs, btrfs, squashfs, nfs** and **vfat**
- A list of currently supported kernel filesystems can be seen at `/proc/filesystems`

You can see a list of the filesystem types currently registered and understood by the currently running **Linux** kernel by doing:

```
$ cat /proc/filesystems
```

```
nodev   sysfs           nodev   bpf             nodev   autofs
nodev   tmpfs           nodev   pipefs          nodev   mqueue
nodev   bdev            nodev   ramfs           nodev   pstore
nodev   proc            nodev   hugetlbfs               fuseblk
nodev   cgroup          nodev   rpc_pipefs      nodev   fuse
nodev   cgroup2         nodev   devpts          nodev   fusectl
nodev   cpuset                  squashfs                ext3
nodev   devtmpfs                vfat                    ext2
nodev   configfs                msdos                   ext4
nodev   debugfs                 exfat                   ntfs3
nodev   tracefs         nodev   nfs             nodev   overlay
nodev   securityfs      nodev   nfs4
nodev   sockfs          nodev   nfsd
```

(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.

© Copyright the Linux Foundation 2023. All rights reserved.

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, devfs, debugfs**
- Network filesystems such as **NFS**, **coda**, **afs**
- etc.

Linux supports many types of filesystems, and more are constantly being added. These may be:

- Native filesystems designed directly for **Linux**, such as **ext4**.
- Filesystems brought over from other operating systems, such as **xfs** and **ntfs**.
- Special filesystems which are not real filesystems, but use the infrastructure to accomplish particular purposes, such as **debugfs**

Linux is structured in a very modular fashion which renders this flexibility which is put to good use.

© Copyright the Linux Foundation 2023. All rights reserved.

19.5 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**, **jfs**, **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.

© Copyright the Linux Foundation 2023. All rights reserved.

19.6 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 behavior, or for implementing particular functions. A partial list:

Table 19.1: **Special Filesystems**

Filesystem	Mount Point	Purpose
rootfs	None	During kernel load, provides an empty root directory.
hugetlbfs	Anywhere	Provides extended page access (2 or 4 MB on **x86**)
bdev	None	Used for block devices.
proc	`/proc`	Pseudo filesystem access to many kernel structures and sub-systems.
sockfs	None	Used by **BSD Sockets**
tmpfs	Anywhere	RAM disk with swapping, resizing.
shm	None	Used by System V IPC Shared Memory.
pipefs	None	Used for pipes.
binfmt_misc	Anywhere	Used by various executable formats.
devpts	`/dev/pts`	Used by **Unix98** pseudo terminals
usbfs	`/proc/bus/usb`	Used by **USB** sub-system for dynamical devices.
sysfs	`/sys`	Used as a **device tree**.
debugfs	`/sys/kernel/debug`	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.

© Copyright the Linux Foundation 2023. All rights reserved.

19.7 Labs

Video Demonstration Resources

using_available_filesystems_demo.mp4

Exercise 19.1: 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
```

```
Filesystem     Type   Size  Used Avail Use% Mounted on
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:

© Copyright the Linux Foundation 2023. All rights reserved.

```
$ df -h | grep ' tmpfs'
```

```
tmpfs           tmpfs     7.8G   38M  7.8G   1% /dev/shm
tmpfs           tmpfs     7.8G   18M  7.8G   1% /run
tmpfs           tmpfs     7.8G     0  7.8G   0% /sys/fs/cgroup
tmpfs           tmpfs     1.6G  1.2M  1.6G   1% /run/user/42
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`.

© Copyright the Linux Foundation 2023. All rights reserved.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 20

Disk Partitioning

20.1 Common Disk Types . . . 300
20.2 Disk Geometry . . . 301
20.3 Partitioning . . . 302
20.4 Partition Tables . . . 304
20.5 Naming Disk Devices . . . 306
20.6 blkid and lsblk . . . 307
20.7 Sizing up partitions . . . 309
20.8 Backing Up and Restoring Partition Tables . . . 310
20.9 Partition table editors . . . 312
20.10 fdisk . . . 313
20.11 Labs . . . 314

Learning Objectives

By the end of this session, you should be able to:

- Describe and contrast the most common types of hard disks and data buses.
- Explain disk geometry and other partitioning concepts.
- Understand how disk devices are named and how to identify their associated device nodes.
- Use utilities such as fdisk, blkid and lsblk.
- Distinguish among and select different partitioning strategies.
- Back up and restore partition tables.

20.1 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 **SSD**s 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 **SSD**s and rotational drives in the same machines, with frequently accessed and performance critical data transfers taking place on the **SSD**s.

© Copyright the Linux Foundation 2023. All rights reserved.

20.2 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 **SSD**s 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, **SSD**s have no moving parts or anything like the above ingredients.

We can see the geometry with **fdisk**:

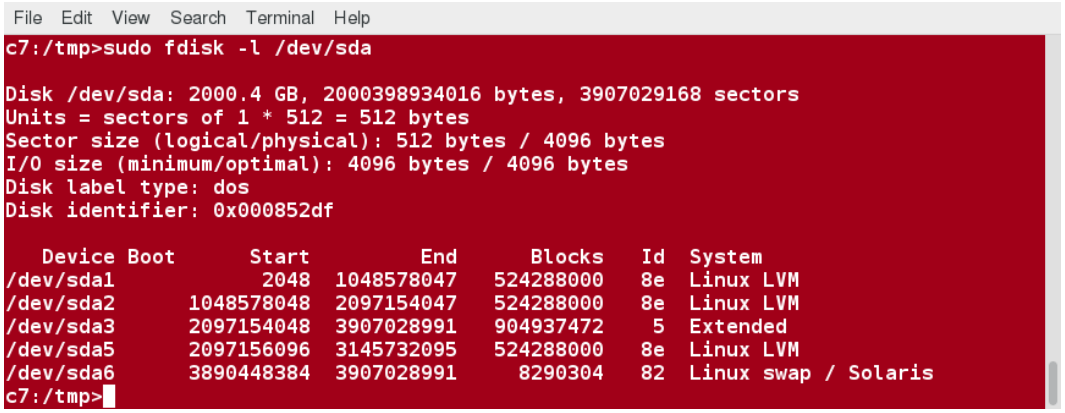


Figure 20.1: **Using fdisk**

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.

© Copyright the Linux Foundation 2023. All rights reserved.

20.3 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.)

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

20.4 Partition Tables

MBR Partition Table

- The Partition table is in the last 64 bytes of the first sector, the 512 bytes of the **MBR** (**M**aster **B**oot **R**ecord)
- 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.

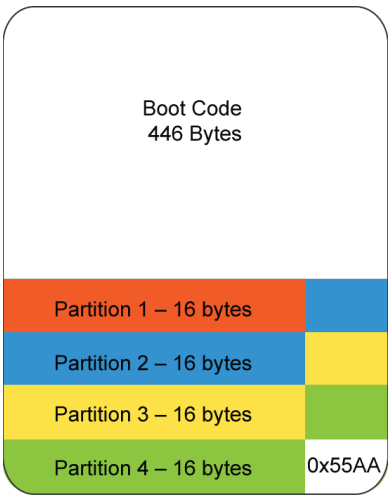


Figure 20.2: **MBR Disk Partition Table**

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`.)

© Copyright the Linux Foundation 2023. All rights reserved.

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

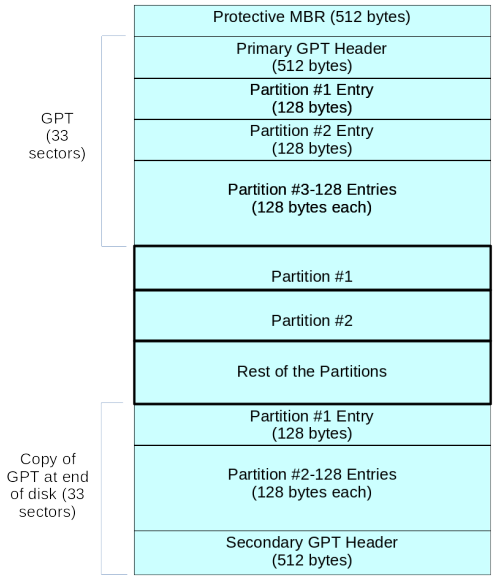


Figure 20.3: **GPT Disk Partition Table**

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.

© Copyright the Linux Foundation 2023. All rights reserved.

20.5 Naming Disk Devices

Naming Disk Devices and Device Nodes

- Device files found in `/dev` directory
- Name format: `xxy[z]`
  - `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.

© Copyright the Linux Foundation 2023. All rights reserved.

20.6 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 metadata (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.

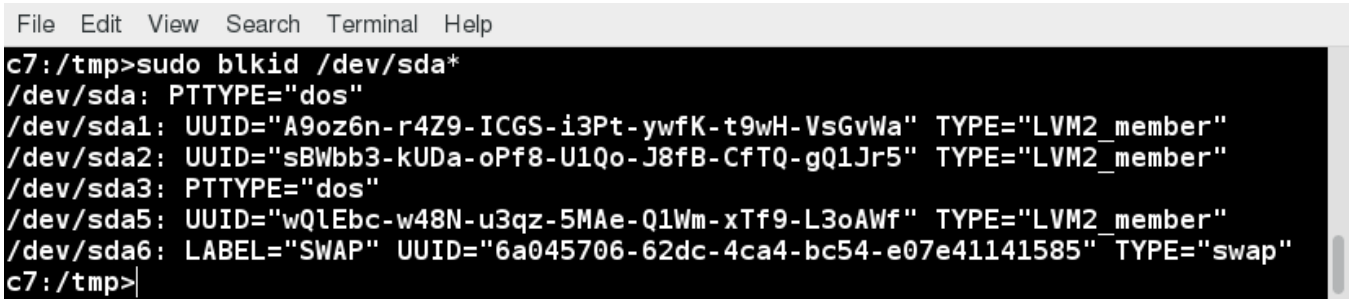


Figure 20.4: **Using blkid**

© Copyright the Linux Foundation 2023. All rights reserved.

lsblk

- Presents block device information in tree format

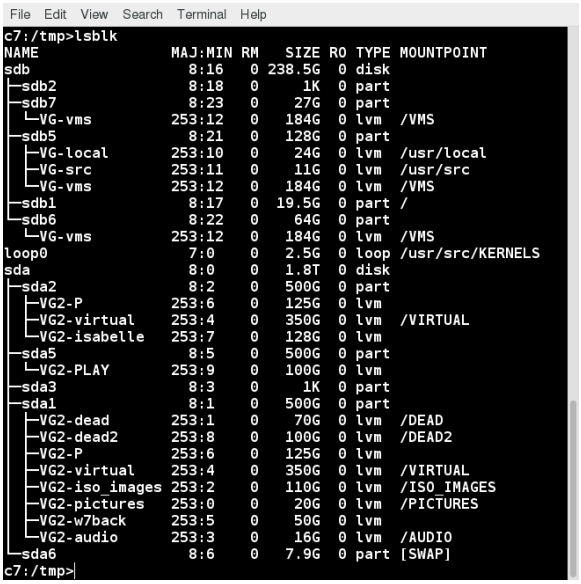


Figure 20.5: **Using lsblk**

```
$ lsblk -h
```

```
Usage:
 lsblk [options] [<device> ...]

List information about block devices.

Options:
 -a, --all            print all devices
 -b, --bytes          print SIZE in bytes rather than in human readable format
 -d, --nodeps         don't print slaves or holders
 -D, --discard        print discard capabilities
 -z, --zoned          print zone model
 -e, --exclude <list> exclude devices by major number (default: RAM disks)
 -f, --fs             output info about filesystems
 -i, --ascii          use ascii characters only
 -I, --include <list> show only devices with specified major numbers
 -J, --json           use JSON output format
 -l, --list           use list format output
 -T, --tree           use tree format output
 -m, --perms          output info about permissions
 -n, --noheadings     don't print headings
 -o, --output <list>  output columns
 -O, --output-all     output all columns
 -p, --paths          print complete device path
 -P, --pairs          use key="value" output format
 -r, --raw            use raw output format
 -s, --inverse        inverse dependencies
 -S, --scsi           output info about SCSI devices
 -t, --topology       output info about topology
 -x, --sort <column>  sort output by <column>

 -h, --help           display this help
 -V, --version        display version

 Available output columns:
 ......
 For more details see lsblk(8).
```


© Copyright the Linux Foundation 2023. All rights reserved.

20.7 Sizing up partitions

Sizing up Partitions

- Many **Linux** systems 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, including **Ubuntu**, default to use of a swap file rather than a partition:
  - Is more flexible (resizing is easier for example)
  - Can be more dangerous, however, if error or bug spreads corruption

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.

On older rotational hard drive media, it may make more sense to have a separate swap partition, but on **SSD**-type media, this is unimportant. However, one still may want to put swap on slower and probably cheaper hardware. This is true whether you use a partition or a file, which is becoming a more prevalent choice.

Note that some distributions are now using (optionally) **zram** (www.kernel.org/doc/html/latest/admin-guide/blockdev/zram.html) which instead of using disk storage for swap, uses compressed memory. This can easily lead to out of memory conditions but in expert hands can improve performance.

© Copyright the Linux Foundation 2023. All rights reserved.

20.8 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
```


© Copyright the Linux Foundation 2023. All rights reserved.

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
    Disk /dev/sda: 1000215216 sectors, 476.9 GiB
    Model: SAMSUNG MZNLN512
    Sector size (logical/physical): 512/512 bytes
    Disk identifier (GUID): DBDBE747-8392-4B62-97AA-6214490073DC
    Partition table holds up to 128 entries
    ....
    Number  Start (sector)    End (sector)  Size       Code  Name
       1            2048          534527   260.0 MiB   EF00  EFI System Partition
       2          534528          567295   16.0 MiB    0C01  Microsoft reserved ...
    ....
```

Note if run on a pure **MBR** system:

```
c8:/tmp>sudo sgdisk -p /dev/sda
    ****************************************************************
    Found invalid GPT and valid MBR; converting MBR to GPT format
    in memory.
    ****************************************************************
    Disk /dev/sda: 500118192 sectors, 238.5 GiB
    Model: Crucial_CT256MX1
    Sector size (logical/physical): 512/4096 bytes
    ....
```


© Copyright the Linux Foundation 2023. All rights reserved.

20.9 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.

© Copyright the Linux Foundation 2023. All rights reserved.

20.10 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.

© Copyright the Linux Foundation 2023. All rights reserved.

20.11 Labs

Video Demonstration Resources

using_fdisk_demo.mp4

Exercise 20.0: 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 to 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 20.1: 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
   ```

   ```
   mke2fs 1.42.9 (28-Dec-2013)
   imagefile is not a block special device.
   Proceed anyway? (y,n) y
   Discarding device blocks: done
   .....
   ```

   Of course you can format with a different filesystem, doing **mkfs.ext3**, **mkfs.vfat**, **mkfs.xfs** etc.

© Copyright the Linux Foundation 2023. All rights reserved.

3. Mount it somewhere:

```
$ mkdir mntpoint
$ sudo mount -o loop imagefile mntpoint
```

   You can now use this to your heart's content, putting files etc. on it.

4. 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 20.2: 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.

1. Run **fdisk** on `imagefile`:

```
$ sudo fdisk -C 130 imagefile
```

```
Device does not contain a recognized partition table
Building a new DOS disk label with disk identifier 0x6280ced3.
Welcome to fdisk (util-linux 2.23.2).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

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.

2. Type `m` to get a list of commands:

```
m
```

```
Command (m for help): m
Command action
a   toggle a bootable flag
b   edit bsd disklabel
c   toggle the dos compatibility flag
d   delete a partition
g   create a new empty GPT partition table
```


© Copyright the Linux Foundation 2023. All rights reserved.

```
G   create an IRIX (SGI) partition table
l   list known partition types
m   print this menu
n   add a new partition
o   create a new empty DOS partition table
p   print the partition table
q   quit without saving changes
s   create a new empty Sun disklabel
t   change a partition's system id
u   change display/entry units
v   verify the partition table
w   write table to disk and exit
x   extra functionality (experts only)

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
```

```
Partition type:
p   primary (0 primary, 0 extended, 4 free)
e   extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-2097151, default 2048):
Using default value 2048
Last sector, +sectors or +size{K,M,G} (2048-2097151, default 2097151): +256M
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
```

```
Partition type:
p   primary (1 primary, 0 extended, 3 free)
e   extended
Select (default p): p
Partition number (2-4, default 2): 2
First sector (526336-2097151, default 526336):
Using default value 526336
Last sector, +sectors or +size{K,M,G} (526336-2097151, default 2097151): +256M
Partition 2 of type Linux and of size 256 MiB is set

Command (m for help): p

Disk imagefile: 1073 MB, 1073741824 bytes, 2097152 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x6280ced3

Device Boot      StartEnd      Blocks   Id  System
imagefile1          2048      526335      262144   83  Linux
imagefile2          526336     1050623      262144   83  Linux
```

5. Write the partition table to disk and exit:

© Copyright the Linux Foundation 2023. All rights reserved.

```
Command (m for help): w
```

```
The partition table has been altered!

Syncing disks.
```

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 20.3: 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
```

```
/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
```

```
/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
```

```
Disk /dev/loop1: 1073 MB, 1073741824 bytes, 2097152 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
```


© Copyright the Linux Foundation 2023. All rights reserved.

```
Disk identifier: 0x00050c11

      Device Boot      Start         End      Blocks   Id  System
/dev/loop1p1               1      500000      250000   83  Linux
/dev/loop1p2          500001     1000000      250000   83  Linux
/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:

```
$ ls -l /dev/loop1*
```

```
brw-rw---- 1 root disk   7, 1 Oct  7 14:54 /dev/loop1
brw-rw---- 1 root disk 259, 0 Oct  7 14:54 /dev/loop1p1
brw-rw---- 1 root disk 259, 3 Oct  7 14:54 /dev/loop1p2
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
```

```
Filesystem               Type      Size  Used Avail Use% Mounted on
/dev/sda1                ext4       29G  8.5G   19G  32% /
....
/dev/loop1p1             ext3      233M  2.1M  219M   1% mnt1
/dev/loop1p2             ext4      233M  2.1M  215M   1% mnt2
/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 20.4: 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.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 21

Filesystem Features: Attributes, Creating, Checking, Usage, Mounting

21.1 Extended Attributes . . . 320
21.2 Creating and formatting filesystems . . . 321
21.3 Troubleshooting Filesystems . . . 322
21.4 Checking and Repairing Filesystems . . . 324
21.5 Filesystem Usage . . . 325
21.6 Disk Usage . . . 326
21.7 Mounting filesystems . . . 327
21.8 NFS . . . 331
21.9 Mounting at Boot and /etc/fstab . . . 332
21.10 automount . . . 334
21.11 Network Block Devices . . . 336
21.12 Labs . . . 340

Learning Objectives

By the end of this session, you should be able to:

- Learn how to use extended file attributes with **lsattr** and **chattr**.
- Create and format filesystems with **mkfs** and **fsck**.
- Troubleshoot filesystem problems.
- Check and fix errors on filesystems.
- Use the utilities **df** and **du** to check usage of disk space.
- Mount and unmount filesystems.
- Use network file systems, such as **NFS**.
- Understand how to **automount** filesystems and make sure they mount at boot.
- Understand and use **Network Block Devices** (**NBD**).

21.1 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** (**ACL**s) 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
```


© Copyright the Linux Foundation 2023. All rights reserved.

21.2 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:

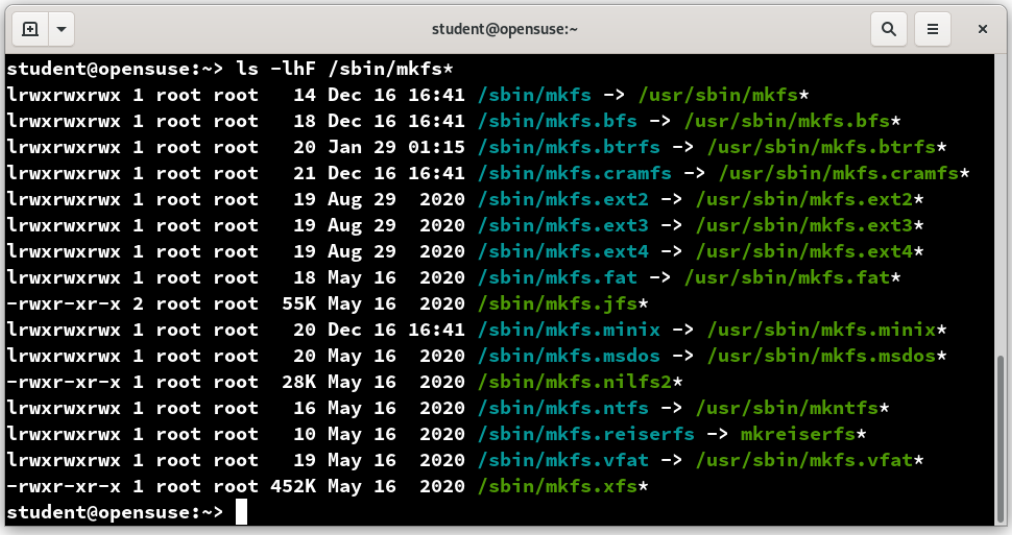


Figure 21.1: **mkfs**

One should look at the **man** page for each of the **mkfs.*** programs to see details.

© Copyright the Linux Foundation 2023. All rights reserved.

21.3 Troubleshooting Filesystems

Checking 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.

© Copyright the Linux Foundation 2023. All rights reserved.

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. thus, 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.

© Copyright the Linux Foundation 2023. All rights reserved.

21.4 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:

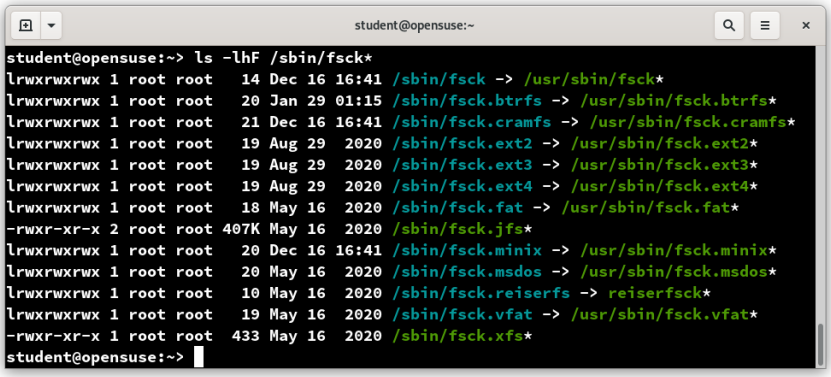


Figure 21.2: **fsck**

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.

© Copyright the Linux Foundation 2023. All rights reserved.

21.5 Filesystem Usage

df: Filesystem Usage

- **df** (**d**isk **f**ree) 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 21.3: **Using df**

© Copyright the Linux Foundation 2023. All rights reserved.
THE LINUX FOUNDATION | Training & Certification

21.6 Disk Usage

du: Disk Usage

- `du` (**d**isk **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 21.4: **Using du**

© Copyright the Linux Foundation 2023. All rights reserved.

21.7 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.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

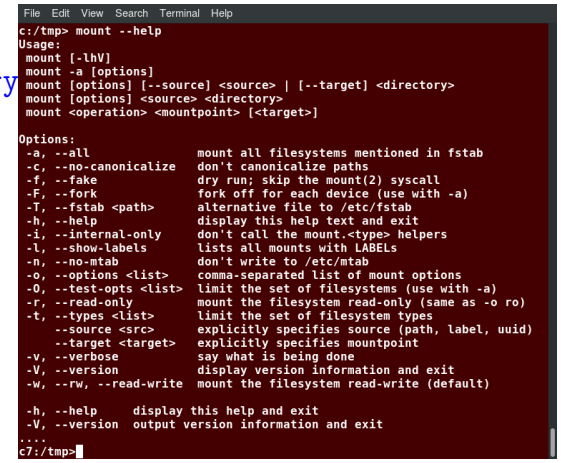


Figure 21.5: **mount**

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 **UUID**s 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. **UUID**s, however, should always be unique and are created when the partitions are created.

© Copyright the Linux Foundation 2023. All rights reserved.

Currently Mounted Filesystems

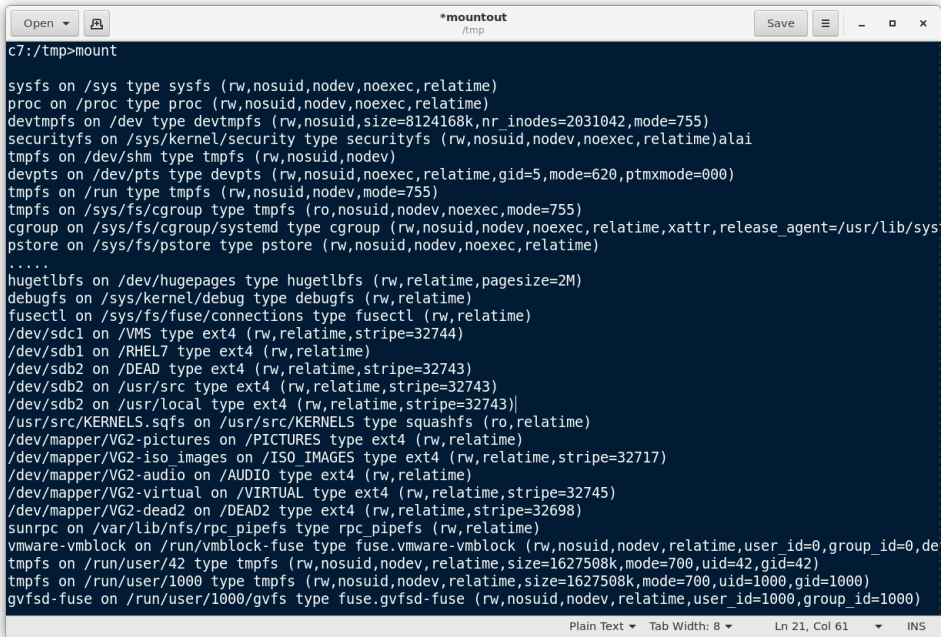


Figure 21.6: **Currently Mounted Filesystems**

Above we see three basic types of filesytems:

- *kernelspace* - backed by a block device

  ext4

- *userspace* - backed by a block device

  vmware-vmblock - a **FUSE** filesystem used to share `VMWare` volumes

  gvfsd-fuse - a **FUSE** filesystem for handling mounts in **GNOME**

- *psuedo* - not backed by a block device (but also generally *kernelspace*)

  sysfs, **procfs**, **devtmpfs**, and **devtmpfs** - providing core functionality

  debugfs, **hugetlbfs**, **cgroup**, and **sunrpc** - providing extra functionality, including network filesystem support

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

21.8 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,wsize=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** (**N**etwork **F**ile **S**ystem).

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** (**A**ndrew **F**ile **S**ystem) and **SMB** (**S**erver **M**essage **B**lock), 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`.

© Copyright the Linux Foundation 2023. All rights reserved.

21.9 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.

© Copyright the Linux Foundation 2023. All rights reserved.

/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 21.7: **/etc/fstab Example**

© Copyright the Linux Foundation 2023. All rights reserved.

21.10 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.

© Copyright the Linux Foundation 2023. All rights reserved.

automount Example

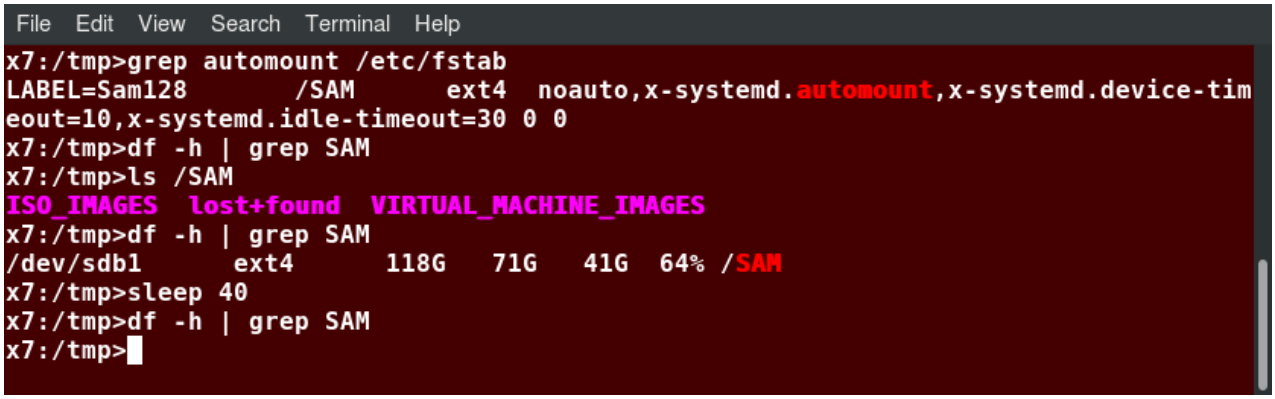


Figure 21.8: **automount Example**

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.

© Copyright the Linux Foundation 2023. All rights reserved.

21.11 Network Block Devices

Introduction to NBD

- Linux protocol designed to export a block device from a source computer (server) to a target (client).
- The **NBD** can use either **Unix sockets** or **TCP/IP** for communication.
- The unit that is exported by the server can be one or more files, image or a block device.
- On the client side the data blob presented by the server is mapped through a **nbd kernel module** and accessed as a block device.
- The client side block devices can be recognized by names like: `/dev/nbd0/`, `/dev/nbd1` etc.
- In its simplest configuration the datastream is not encrypted, however, encryption is available and part of the NBD specification.

Some additional information and reference documents can be found at the following locations:

The **NBD** protocol information:

https://github.com/NetworkBlockDevice/nbd/blob/master/doc/proto.md

Userland client and server application for **NBD**:

https://github.com/NetworkBlockDevice/nbd

© Copyright the Linux Foundation 2023. All rights reserved.

Network Block Device

To configure a **nbd** client/server pair, the general steps are:

- Define something to export on the server
- Define the item to be shared to the server
- Connect the client
- The device can be partitioned and formatted like any other block device by using the **/dev/nbd0, /dev/nbd1** devices
- Almost any file system type can be used with the **nbd** devices once partitioned

The steps we are going to use are:

1. Use **dd** to create an empty file.
2. Define the item to be shared by the server in a configuration file.
3. Activate the **nbd** kernel module.
4. Query the server with the client using the export name, IP address and port
5. Associate the local `/dev/nbd` block driver with the server with the **nbd-client** command.
6. Use `fdisk` to partition the nbd.
7. Add a filesystem to the nbd and mount it.

© Copyright the Linux Foundation 2023. All rights reserved.

NBD user utilities

There are several **nbd** clients and server packages available, including:

- **nbdkit**: **CentOS**, **Fedora**, **Debian**, **Ubuntu**
- **nbd-client** and **nbd-server**: **Ubuntu**, **Debian**
- **nbd** (from **GitHub**): **CentOS**, **Fedora**, **Debian**, **Ubuntu**
- **xNBD-client** and **xNBD-server**: **Debian**
- **qemu-img**: **CentOS**

In general the clients and servers can be mixed and matched, so careful testing in your use case is recommended.

Which package are we going to use?

In our lab environment we are going to use the **nbd** package from **GitHub**. It is not the most complex of the packages but it runs on almost any **Linux** version.

An example of the user and administrator utilities for @ubuntu 22.04:

- **nbd-server.conf** is an example of a configuration file for the server containing:
  - IP address and port to listen for connections
  - storage device to export as a disk
  - some optional control elements
- **nbd-server** is the server side component to answer connection requests and communication
- **nbd-server man page** contains the specifics for configuration of the server. This information may vary between distributions.
- **nbd-client** is used to query the server and make the connection to the server.
- **nbd-client man page** contains client related information to make the connection to the server.

These utilities may have different names and include different functions depending on how they are packaged by the distributions.

© Copyright the Linux Foundation 2023. All rights reserved.

Network Block Device Example

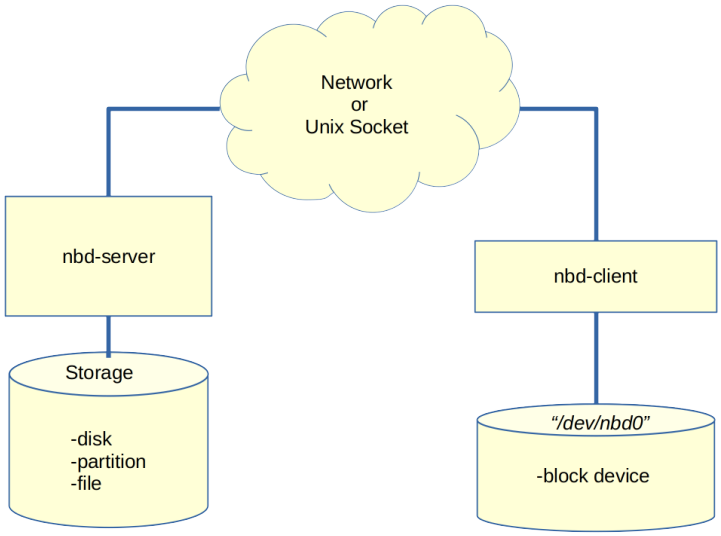


Figure 21.9: **Network Block Device**

Some example commands for the clients and server.
The server was **CentOS-8-Stream** using the **nbd** package from **GitHub**.
The client was **CentOS-8-Stream** using the **nbd** package from **GitHub**.

- Ensure the nbd kernel modules are loaded.

```
$ sudo  modprobe -i nbd
```

- Connect the export foo on 192.168.242.160 to the local device /dev/nbd10

```
$ sudo  nbd-client -N foo 192.168.242.160 /dev/nbd10
```

An example of some commands from a **Ubuntu** installation:

- Start the nbd server process

```
$ sudo nbd-server -C nbd-server.conf
```

- List the exports on the server from the client

```
$ sudo nbd-client -l 127.0.0.1 10042
```

- Connect the export "foo" to local device /dev/nbd0

```
$ sudo nbd-client -N foo 127.0.0.1 10042 /dev/nbd0
```


© Copyright the Linux Foundation 2023. All rights reserved.

21.12 Labs

Video Demonstration Resources

using_filesystems_demo.mp4

Exercise 21.1: 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 21.1

1.
```
$ cd /tmp
$ touch appendit
$ ls -l appendit
```
```
-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
```
```
chattr: Operation not permitted while setting flags on appendit
```

5.
```
$ sudo chattr +a appendit
$ lsattr appendit
```
```
-----a-------e-- appendit
```

6.
```
$ cat /etc/passwd > appendit
```


© Copyright the Linux Foundation 2023. All rights reserved.

```
bash: appendit: Operation not permitted
```

7.
```
$ sudo su
$ cat /etc/passwd > appendit
```

```
bash: appendit: Operation not permitted
```

```
$ exit
```

8.
```
$ cat /etc/passwd >> /tmp/appendit
$ cat appendit
```

9.
```
$ sudo chattr +i appendit
$ lsattr appendit
```

```
----ia-------e- appendit
```

10.
```
$ echo hello >> appendit
```

```
-bash: appendit: Permission denied
```

```
$ mv appendit appendit.rename
```

```
mv: cannot move `appendit' to `appendit.rename': Operation not permitted
```

```
$ ln appendit appendit.hardlink
```

```
ln: creating hard link `appendit.hardlink' => `appendit': Operation not permitted
```

```
$ rm -f appendit
```

```
rm: cannot remove `appendit': Operation not permitted
```

```
$ sudo su
$ echo hello >> appendit
```

```
-bash: appendit: Permission denied
```

```
$ mv appendit appendit.rename
```

```
mv: cannot move `appendit' to `appendit.rename': Operation not permitted
```

```
$ ln appendit appendit.hardlink
```

```
ln: creating hard link `appendit.hardlink' => `appendit': Operation not permitted
```


© Copyright the Linux Foundation 2023. All rights reserved.

```
$ rm -f appendit
```

```
rm: cannot remove `appendit': Operation not permitted
```

```
$ exit
```

11.
```
$ sudo su
$ lsattr appendit
```

```
----ia-------e- appendit
```

```
$ chattr -ia appendit
$ rm appendit
```

```
rm: remove regular file `appendit'? y
```

```
$ ls appendit
```

```
ls: cannot access appendit: No such file or directory
```

Exercise 21.2: 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 21.2

© Copyright the Linux Foundation 2023. All rights reserved.

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
   ```

   ```
   .....
   w
   $ partprobe -s
   ```

   Sometimes the **partprobe** won't work, and to be sure the system knows about the new partition you have to reboot.

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
   ```


© Copyright the Linux Foundation 2023. All rights reserved.

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:

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?

Exercise 21.3: Configure and test NBD

Overview

In this lab we are going to configure and test **NBD**, communicating via the **loopback** device. Both client and server will be running on the same system.

For the **NBD** export, an empty file will be used.

The /opt directory is to be used for the exported device and configuration files. Production environments may prefer different locations.

© Copyright the Linux Foundation 2023. All rights reserved.

CentOS-stream-8 issue

At creation time **CentOS-8-Stream** is not working correctly.

Please use **CentOS-Stream-9** or **Fedora**. Waiting for resolution.

1. First step:

On Red Hat, Centos, or Fedora

Prepare a directory to house the source code downloaded with **git**. Clone, build the **NBD** programs and install.

On Debian, Ubuntu, or Linux Mint

Install the **NBD** client and server packages.

2. Create a `nbd-server.conf` file in the `/opt` directory. A minimal, no comments file is sufficient. (A more interesting configuration file can be found in the `SOLUTIONS` directory.
3. Create two files for exporting and set the ownership to `student`.
4. Start the **nbd-server**.
5. Verify the nbd server responds to a query from the client.
6. Install the **nbd** kernel module, verify the block devices are present.
7. Connect the server supplied image, using the ip address, port to the local block device.
8. Wipe out any existing information on the **NBD**.
9. Using the **NBD**, create a **GPT**.
10. Create a partition on the **NBD**.
11. Add an **ext3** filesystem to the **NBD** and test.

Solution 21.3

1. First step:

On Red Hat, Centos, or Fedora

```
$ mkdir ~/src
$ cd ~/src
$ git clone https://github.com/NetworkBlockDevice/nbd.git
$ cd nbd/
```

The packages `docbook-utils` and `autoconf-archive` may not be installed on your system, install them now:

```
$ sudo dnf install docbook-utils
$ sudo dnf install autoconf-archive
```

Build the programs:

```
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
```


© Copyright the Linux Foundation 2023. All rights reserved.

On Debian, Ubuntu, or Linux Mint

The **Debian**, **Ubuntu**, or **Linux Mint**have deb packages available, install the **NBD** utilities.

```
$ sudo apt install nbd-client nbd-server
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  nbd-client nbd-server
0 upgraded, 2 newly installed, 0 to remove and 43 not upgraded.
\ldots output truncated
```

2. Create the **nbd-server** configuration file in the `/opt` directory. **sudo** will be required to edit files in the `/opt` directory.

```
$ sudo /opt/nbd-server.conf
```

/opt/nbd-server.conf

```
[generic]
    allowlist = 1
    listenaddr = 0.0.0.0
    port = 10042
[foo]
    exportname = /opt/dsk1
    readonly = false
[bar]
    exportname = /opt/dsk2
```

3.
```
sudo dd if=/dev/zero of=/opt/dsk1 status=progress bs=100M count=5
sudo dd if=/dev/zero of=/opt/dsk2 status=progress bs=100M count=5
sudo chmod 777 /opt
sudo chown student.student /opt/*
```

4.
```
$ sudo nbd-server -C /opt/nbd-server.conf
```

5.
```
$ sudo nbd-client -l 127.0.0.1 -p 10042
```

```
Negotiation: ..
foo
bar
```

6.
```
$ sudo modprobe -i nbd
$ ls /dev/nbd*
```

```
/dev/nbd0   /dev/nbd11  /dev/nbd14  /dev/nbd3  /dev/nbd6  /dev/nbd9
/dev/nbd1   /dev/nbd12  /dev/nbd15  /dev/nbd4  /dev/nbd7
/dev/nbd10  /dev/nbd13  /dev/nbd2   /dev/nbd5  /dev/nbd8
```

7.
```
$ sudo nbd-client -N foo  127.0.0.1 10042 /dev/nbd0
```

8. Wipe out the NBD:

© Copyright the Linux Foundation 2023. All rights reserved.

```
$ sudo dd if=/dev/zero of=/dev/nbd0 bs=1M count=5
```

```
5+0 records in
5+0 records out
5242880 bytes (5.2 MB, 5.0 MiB) copied, 0.00161564 s, 3.2 GB/s
```

9. Create a GPT label on the NBD:

```
$ sudo su -c "echo 'label: gpt' | sfdisk /dev/nbd0"
```

```
Checking that no-one is using this disk right now ... OK

Disk /dev/nbd0: 4 GiB, 4294967296 bytes, 8388608 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes

>>> Script header accepted.
>>> Done.
Created a new GPT disklabel (GUID: 000C20C1-6929-424F-93E1-28319DE1FF1F).

New situation:
Disklabel type: gpt
Disk identifier: 000C20C1-6929-424F-93E1-28319DE1FF1F

The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```

10. Add a partition to the NBD:

```
$ sudo su -c "echo ';' | sfdisk /dev/nbd0"
```

```
Checking that no-one is using this disk right now ... OK

Disk /dev/nbd0: 4 GiB, 4294967296 bytes, 8388608 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 000C20C1-6929-424F-93E1-28319DE1FF1F

Old situation:

>>> Created a new GPT disklabel (GUID: EFFC087B-A15C-7D42-809C-4BDED58EE238).
/dev/nbd0p1: Created a new partition 1 of type 'Linux filesystem' and of size 4 GiB.
/dev/nbd0p2: Done.

New situation:
Disklabel type: gpt
Disk identifier: EFFC087B-A15C-7D42-809C-4BDED58EE238

Device      Start     End Sectors Size Type
/dev/nbd0p1  2048 8388574 8386527   4G Linux filesystem

The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```


© Copyright the Linux Foundation 2023. All rights reserved.

11. Add an **ext3** filesystem to the **NBD** and test:

```
$ sudo mkfs.ext3 -L nbd-foo /dev/nbd0
```

```
mke2fs 1.46.5 (30-Dec-2021)
Creating filesystem with 128000 4k blocks and 128000 inodes
Filesystem UUID: 6a01ff5a-9ad5-4f43-8050-9f2b8df26c14
Superblock backups stored on blocks:
        32768, 98304

Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done
```

```
$ sudo mount LABEL=nbd-foo /mnt
$ sudo touch /mnt/file1
$ ls -l /mnt
```

```
total 16
-rw-r--r-- 1 root root     0 Jun  9 10:58 file1
drwx------ 2 root root 16384 Jun  9 10:56 lost+found
```


© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 22

The Ext4 Filesystems

22.1 ext4 Features 350
22.2 ext4 Layout and Superblock and Block Groups 351
22.3 dumpe2fs 354
22.4 tune2fs 355
22.5 Labs 356

Learning Objectives

By the end of this session, you should be able to:

- Describe the main features of the **ext4** filesystem and how it is laid out on disk.
- Explain the concepts of block groups, superblock, data blocks and inodes.
- Use the **dumpe2fs** and **tune2fs** utilities.

22.1 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.

© Copyright the Linux Foundation 2023. All rights reserved.

22.2 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.

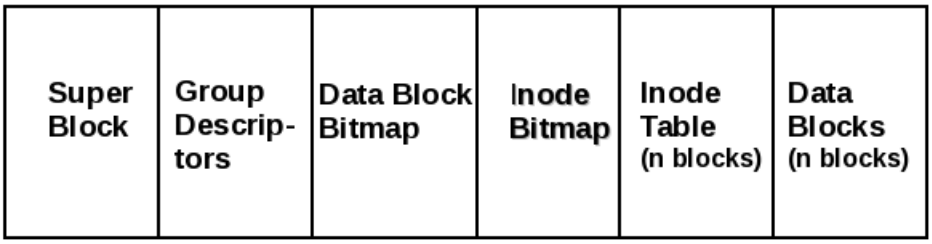


Figure 22.1: **ext[3,4] Filesystem Layout**

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 is followed by the data block bitmap, the inode bitmap, the inode table, and the data blocks.

© Copyright the Linux Foundation 2023. All rights reserved.

Block Groups

- Duplicate Superblocks and Group Descriptors help in filesystem recovery

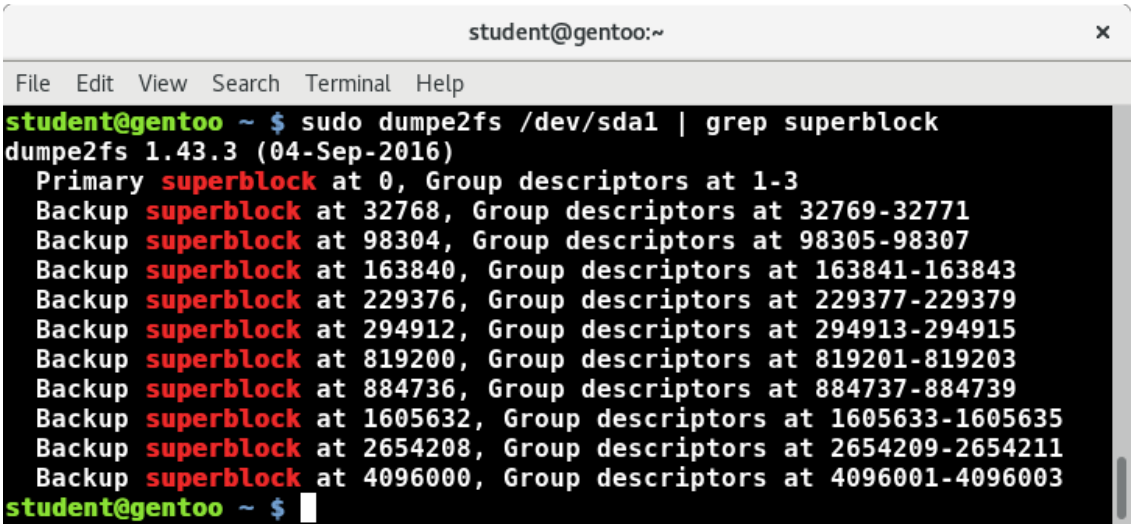


Figure 22.2: **Block Groups**

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.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

22.3 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:          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:       Linuxxf .
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
....
```


© Copyright the Linux Foundation 2023. All rights reserved.

22.4 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:          /VMS
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
....
```


© Copyright the Linux Foundation 2023. All rights reserved.

22.5 Labs

Exercise 22.1: 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
```

```
Usage   : e4defrag [-v] file...| directory...| device...
        : 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
```

```
<Fragmented files>                          now/best       size/ext
1. /var/log/lastlog                           5/1             9 KB
2. /var/log/sa/sa24                           3/1            80 KB
3. /var/log/rhsm/rhsm.log                     2/1           142 KB
4. /var/log/messages                          2/1          4590 KB
5. /var/log/Xorg.1.log.old                    1/1            36 KB

 Total/best extents                          120/112
 Average size per extent                     220 KB
```


© Copyright the Linux Foundation 2023. All rights reserved.

```
Fragmentation score                          1
[0-30 no problem: 31-55 a little bit fragmented: 56- needs defrag]
This directory (/var/log) does not need defragmentation.
Done.
```

```
$ sudo e4defrag  /var/log
```

```
ext4 defragmentation for directory(/var/log)
[2/152]/var/log/Xorg.2.log:     100%    [ OK ]
[3/152]/var/log/Xorg.0.log.old: 100%    [ OK ]
[4/152]/var/log/messages-20141019.gz:   100%    [ OK ]
[5/152]/var/log/boot.log:       100%    [ OK ]
[7/152]/var/log/cups/page_log-20140924.gz:      100%    [ OK ]
[8/152]/var/log/cups/access_log-20141019.gz:    100%    [ OK ]
[9/152]/var/log/cups/access_log:        100%    [ OK ]
[10/152]/var/log/cups/error_log-20141018.gz:    100%    [ OK ]
[11/152]/var/log/cups/error_log-20141019.gz:    100%    [ OK ]
[12/152]/var/log/cups/access_log-20141018.gz:   100%    [ OK ]
[14/152]/var/log/cups/page_log-20141018.gz:     100%    [ OK ]
...
[152/152]/var/log/Xorg.1.log.old:       100%    [ OK ]

        Success:                        [ 112/152 ]
        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 22.2: 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 22.2

© Copyright the Linux Foundation 2023. All rights reserved.

1. `$ dumpe2fs imagefile > dumpe2fs-output-initial`

```
dumpe2fs 1.42.9 (28-Dec-2013)
```

`$ grep -i  -e "Mount count" -e "Check interval" -e "Block Count" dumpe2fs-output-initial`

```
Block count:              262144
Reserved block count:     13107
Mount count:              0
Maximum mount count:      -1
Check interval:           0 (<none>)
```

2. `$ tune2fs -c 30 imagefile`

```
tune2fs 1.42.9 (28-Dec-2013)
Setting maximal mount count to 30
```

`$ tune2fs -i 3w imagefile`

```
tune2fs 1.42.9 (28-Dec-2013)
Setting interval between checks to 1814400 seconds
```

`$ tune2fs -m 10 imagefile`

```
tune2fs 1.42.9 (28-Dec-2013)
Setting reserved blocks percentage to 10% (26214 blocks)
```

3. `$ dumpe2fs imagefile > dumpe2fs-output-final`

```
dumpe2fs 1.42.9 (28-Dec-2013)
```

`$ grep -i  -e "Mount count" -e "Check interval" -e "Block Count" dumpe2fs-output-final`

```
Block count:              262144
Reserved block count:     26214
Mount count:              0
Maximum mount count:      30
Check interval:           1814400 (3 weeks)
```

`$ diff dumpe2fs-output-initial dumpe2fs-output-final`

```
14c14
< Reserved block count:     13107
---
> Reserved block count:     26214
29c29
< Last write time:          Wed Oct 26 14:26:19 2016
---
> Last write time:          Wed Oct 26 14:26:20 2016
31c31
< Maximum mount count:      -1
---
```


© Copyright the Linux Foundation 2023. All rights reserved.

```
> Maximum mount count:      30
33c33,34
< Check interval:           0 (<none>)
---
> Check interval:           1814400 (3 weeks)
> Next check after:         Wed Nov 16 13:26:16 2016
```


© Copyright the Linux Foundation 2023. All rights reserved.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 23

Logical Volume Management (LVM)

23.1 Logical Volume Management (LVM) 362
23.2 Volumes and Volume Groups 363
23.3 Working with Logical Volumes 364
23.4 Resizing Logical Volumes 367
23.5 LVM Snapshots ** 368
23.6 Labs 369

Learning Objectives

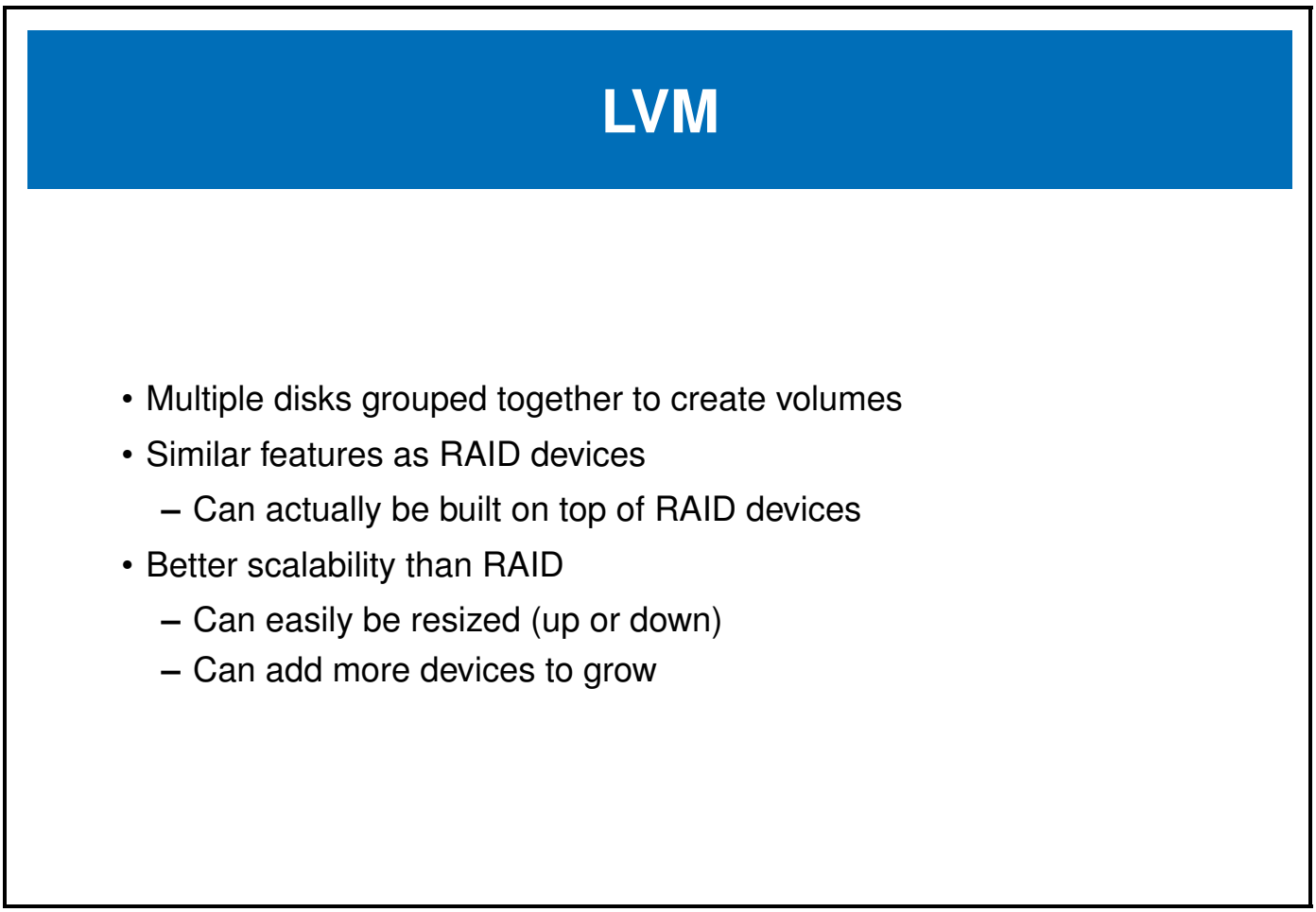
By the end of this session, you should be able to:

- Explain the concepts behind **LVM**.
- Understand the use of logical volumes.
- Create logical volumes.
- Display logical volumes.
- Resize logical volumes.
- Use LVM snapshots.

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 (**L**ogical **V**olume **M**anagement) 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 to 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. 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.

© Copyright the Linux Foundation 2023. All rights reserved.

23.2 Volumes and Volume Groups

Volumes and Volume Groups

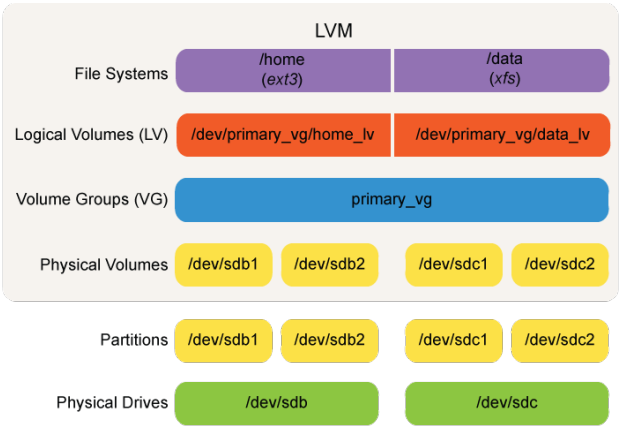


Figure 23.1: **LVM components**

- 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.

© Copyright the Linux Foundation 2023. All rights reserved.

23.3 Working with Logical Volumes

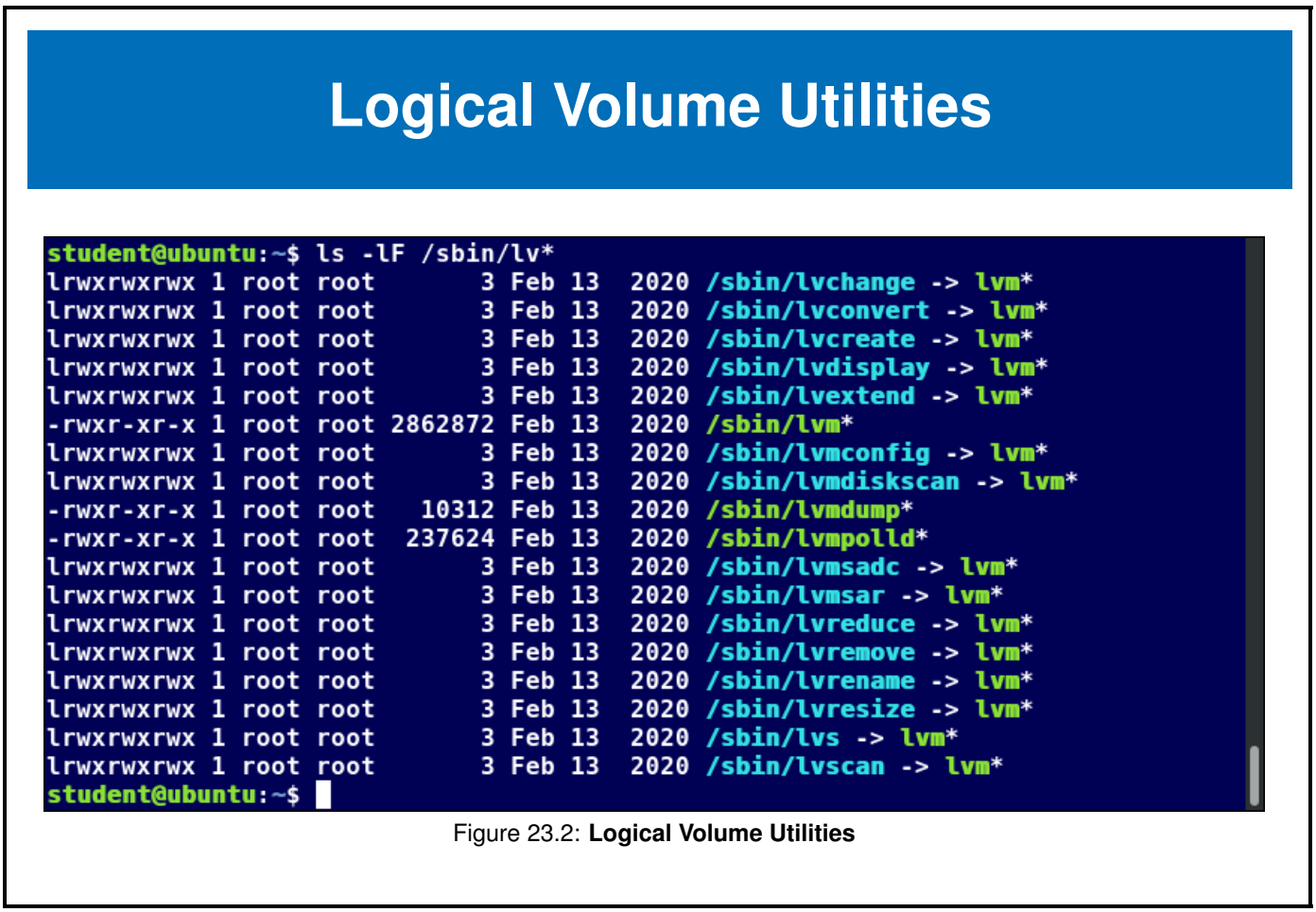


Figure 23.2: **Logical Volume Utilities**

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**

© Copyright the Linux Foundation 2023. All rights reserved.

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.

lvdisplay 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.

© Copyright the Linux Foundation 2023. All rights reserved.

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 the 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**

© Copyright the Linux Foundation 2023. All rights reserved.
THE LINUX FOUNDATION Training & Certification

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:

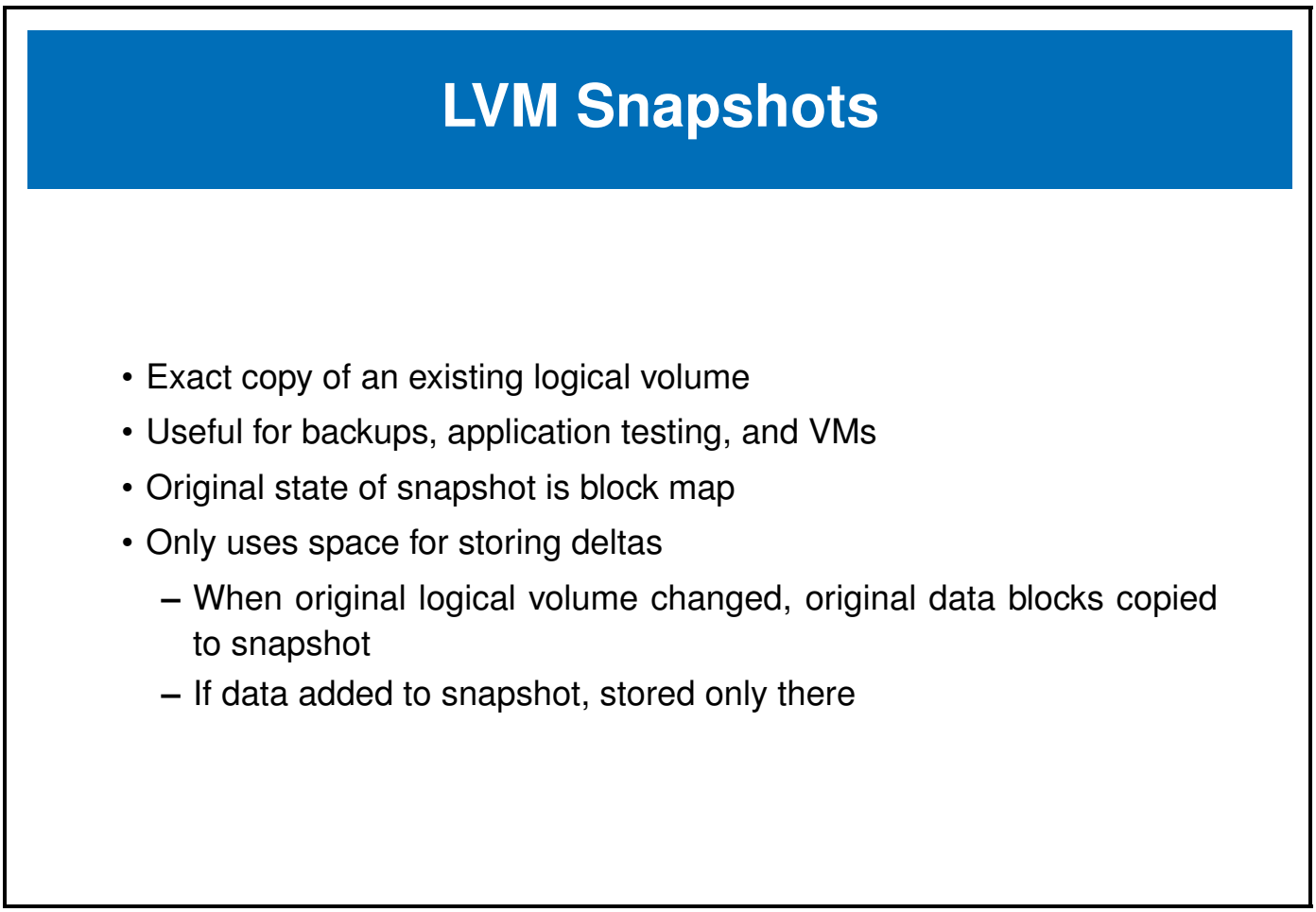
```
$ sudo lvresize -r -L 200M /dev/vg/mylvm
```

You can also reduce a volume group as in:

```
$ sudo pvmove /dev/sdc1
$ sudo vgreduce vg /dev/sdc1
```


© Copyright the Linux Foundation 2023. All rights reserved.

23.5 LVM Snapshots **



LVM Snapshots create an exact copy of an existing logical volume.

They are useful for backups, application testing, and deploying **VM**s (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.

© Copyright the Linux Foundation 2023. All rights reserved.

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 pvdisplay
   ```

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:

© Copyright the Linux Foundation 2023. All rights reserved.
   ```

in /etc/fstab

```
/dev/myvg/mylvm /mylvm ext4 defaults 0 0
```

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
```


© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 24

Kernel Services and Configuration

24.1 Kernel Overview 372
24.2 Kernel Boot Parameters 373
24.3 Kernel Command Line 374
24.4 Boot Process Failures 375
24.5 sysctl 376
24.6 Labs 377

Learning Objectives

By the end of this session, you should be able to:

- Grasp the main responsibilities the kernel must fulfill and how it achieves them.
- Explain what parameters can be set on the kernel command line and how to make them effective either for just one system boot, or persistently.
- Troubleshoot boot process failures.
- Know where to find detailed documentation on these parameters.
- Use **sysctl** to set kernel parameters either after the system starts, or persistently across system reboots.

24.1 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**.

© Copyright the Linux Foundation 2023. All rights reserved.

24.2 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 rhgb quiet crashkernel=512M
```

- No intentionally secret, hidden, parameters
- Enumerated in `kernel-parameters.txt` in the kernel source or on line at https://kernel.org/doc/html/latest/admin-guide/kernel-parameters.html.

Kernel options are placed at the end of the kernel line and are separated by spaces. An example of kernel boot parameters:

```
linux /boot/vmlinuz-5.19.0 root=/dev/sda5 ro crashkernel=512M quiet selinux=0
```

- `root`: root filesystem (can be in the form of `root=UUID=...` or `root=/dev/sda5` or `root=LABEL=CentOS9` etc.
- `ro`: mounts root device read-only on boot
- `crashkernel=512M`: how much memory to set aside for **kernel crashdumps** through the **kdump** facility.
- `quiet`: disables most log messages
- `selinux=0` disables **SELinux**

See **man bootparam** and `kernel-parameters.txt` in the on-machine kernel documentation, which can be found either in the kernel source, or in a package with a name like **kernel-doc**. Even easier, look online at https://kernel.org/doc/html/latest/admin-guide/kernel-parameters.html.

Note that **Linux** distributions often add parameters that are particular to that distribution, such as `rhgb` in the example above, which stands for **Red Hat Graphical Boot**.

© Copyright the Linux Foundation 2023. All rights reserved.

24.3 Kernel Command Line

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-5.19.0 root=UUID=7ef4e747-afae-48e3-90b4-9be8be8d0258 ro quiet

linuxefi /boot/vmlinuz-5.2.9 root=UUID=77461ee7-c34a-4c5f-b0bc-29f4feecc743 ro crashkernel=auto rhgb
↪  quiet crashkernel=384M-:128M
```

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
```

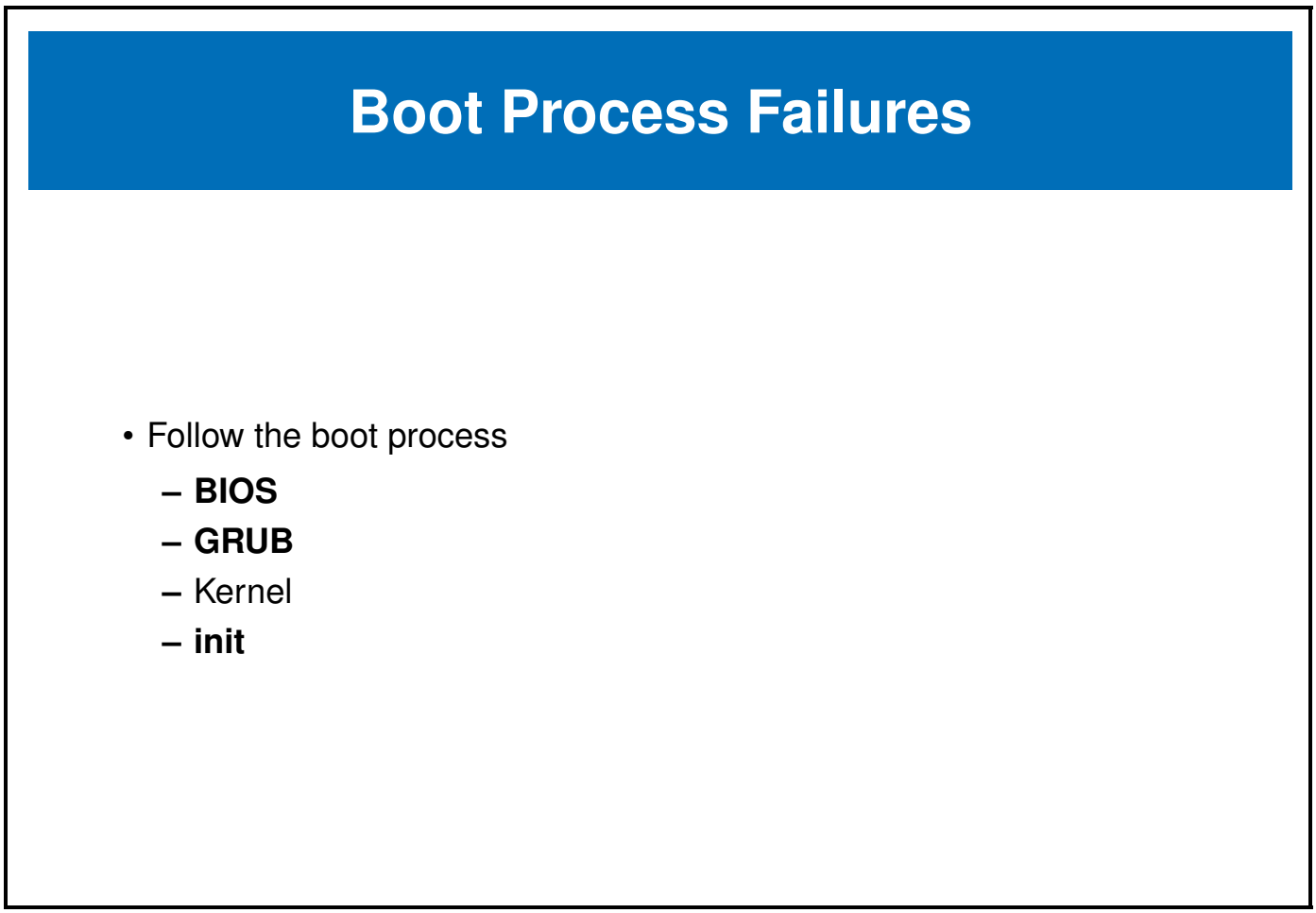
```
BOOT_IMAGE=(hd0,msdos2)/boot/vmlinuz-5.19.0 root=UUID=7f7221b8-60d8-41b9-b643-dfcc80527c37 ro rhgb quiet
↪  crashkernel=512M
```

BLSFG Changes things

Fedora and **RHEL/CentOS** use the **BLSFG** (Boot Loader Specification Configuration) which alters where the kernel command line is set. You should now look in either `/boot/grub2/grubenv` for that information, or the files in `/boot/loader/entries`.

© Copyright the Linux Foundation 2023. All rights reserved.

24.4 Boot Process Failures



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).

© Copyright the Linux Foundation 2023. All rights reserved.

24.5 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

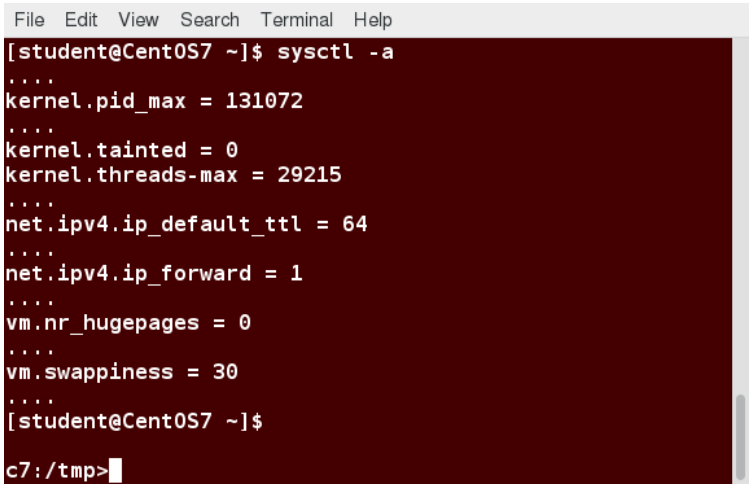


Figure 24.1: **sysctl**

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 self-documenting original file (/etc/sysctl.conf) is still supported.

© Copyright the Linux Foundation 2023. All rights reserved.

24.6 Labs

Video Demonstration Resources

using_sysctl_demo.mp4

Exercise 24.1: 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 in 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 24.1

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 24.2: 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.)

© Copyright the Linux Foundation 2023. All rights reserved.

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 24.2

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] 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 &
```

```
[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.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 25

Kernel Modules

25.1 Kernel Modules 380
25.2 Module Utilities 382
25.3 modinfo 384
25.4 Module Configuration 385
25.5 Labs 386

Learning Objectives

By the end of this session, you should be able to:

- List the advantages of utilizing kernel modules.
- Use **insmod**, **rmmod** and **modprobe** to load and unload kernel modules.
- Use **modinfo** to find out information about kernel modules.
- Learn how to configure modules.

25.1 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.

© Copyright the Linux Foundation 2023. All rights reserved.

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 25.1: **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.

© Copyright the Linux Foundation 2023. All rights reserved.

25.2 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
```


© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

25.3 modinfo

modinfo Example

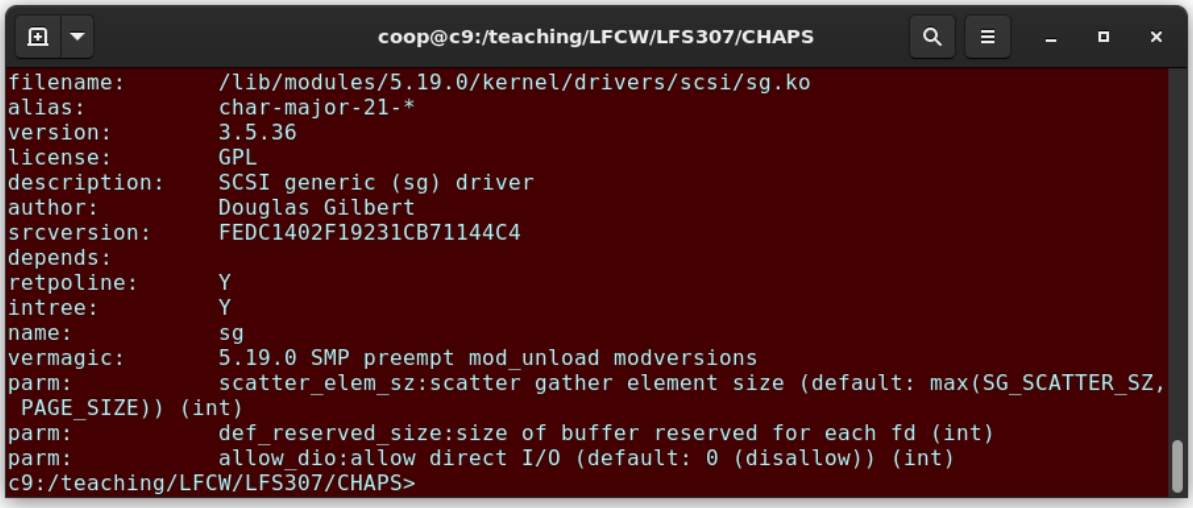


Figure 25.2: **modinfo**

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/sg` and some if not all parameters can be read and/or written under `/sys/module/sg/parameters`. We will show how to set them next.

Many modules can be loaded while specifying parameter values, such as in:

```
$ sudo /sbin/insmod <pathto>/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
```


© Copyright the Linux Foundation 2023. All rights reserved.

25.4 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.

© Copyright the Linux Foundation 2023. All rights reserved.

25.5 Labs

Video Demonstration Resources

using_kernel_modules.mp4

Exercise 25.1: 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 25.1

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`

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 26

Devices and udev

26.1 udev and Device Management 388
26.2 Device Nodes 389
26.3 Rules 392
26.4 Labs 395

Learning Objectives

By the end of this session, you should be able to:

- Explain the role of device nodes and how they use major and minor numbers.
- Understand the need for the udev method and list its key components.
- Describe how the udev device manager functions.
- Identify udev rule files and learn how to create custom rules.

26.1 udev and Device Management

udev

- **udev** stands for **U**ser **Dev**ice 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).

© Copyright the Linux Foundation 2023. All rights reserved.

26.2 Device Nodes

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
```

```
total 0
crw------- 1 coop audio      14,   4 Jul  9 01:54 audio
crw------- 1 root root       10,  62 Jul  9 01:54 autofs
lrwxrwxrwx 1 root root             4 Jul  9 01:54 cdrom -> scd0
lrwxrwxrwx 1 root root             4 Jul  9 01:54 cdrw -> scd0
crw------- 1 coop root        5,   1 Jul  9 06:54 console
....
lrwxrwxrwx 1 root root             4 Jul  9 01:54 dvd -> scd0
lrwxrwxrwx 1 root root             4 Jul  9 01:54 dvdwriter -> scd0
....
brw-r----- 1 root disk        8,   0 Jul  9 01:53 sda
brw-r----- 1 root disk        8,   1 Jul  9 01:53 sda1
brw-r----- 1 root disk        8,   2 Jul  9 06:54 sda2
....
brw-r----- 1 root disk        8,  16 Jul  9 01:53 sdb
brw-r----- 1 root disk        8,  17 Jul  9 01:53 sdb1
brw-r----- 1 root disk        8,  18 Jul  9 01:53 sdb2
....
crw-rw-rw- 1 root tty         5,   0 Jul  9 01:54 tty
crw-rw---- 1 root root        4,   0 Jul  9 14:54 tty0
crw------- 1 root root        4,   1 Jul  9 06:54 tty1
cr--r--r-- 1 root root        1,   9 Jul  9 01:53 urandom
....
crw-rw-rw- 1 root root        1,   5 Jul  9 01:54 zero
```


© Copyright the Linux Foundation 2023. All rights reserved.

Device Nodes Provide Hardware Access

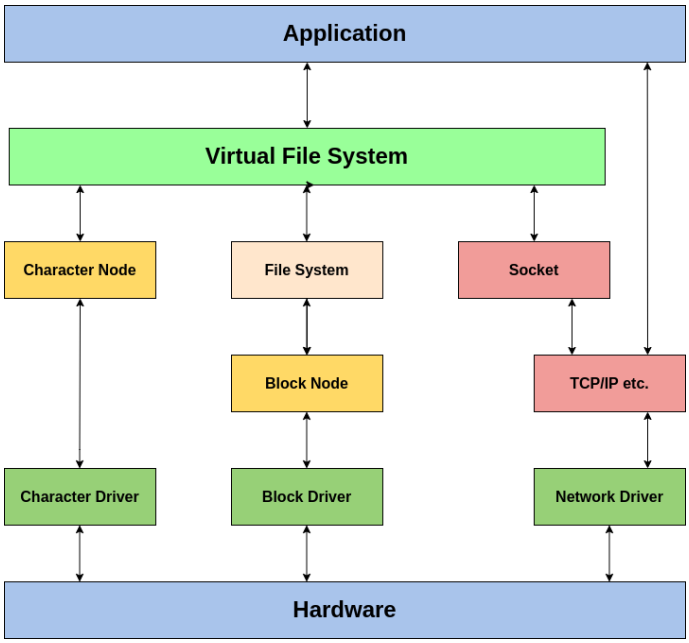


Figure 26.1: **Device Nodes**

In addition to providing some required (statically created outside of udev) **pseudo-devices** including `/dev/null` and `/dev/zero`, the **device nodes** present in `/dev` are the primary or first interaction with the hardware on a machine.

Some common devices are:

- /dev/sd* – devices appearing as hard drives
- /dev/ttyS* – serial ports, often used as consoles
- /dev/snd/* – various sound devices

© Copyright the Linux Foundation 2023. All rights reserved.

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

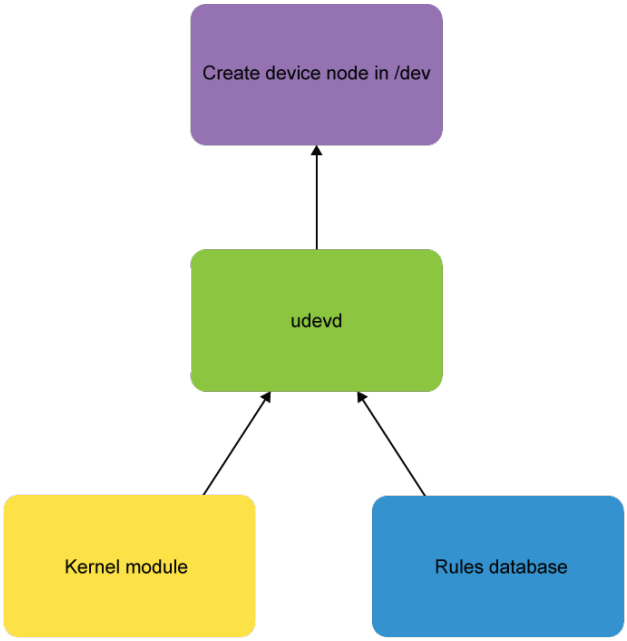


Figure 26.2: **udev Rules**

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.

© Copyright the Linux Foundation 2023. All rights reserved.

26.3 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
```

```
SUBSYSTEM=="usb", ATTR{idVendor}=="2687", ATTR{idProduct}=="fb01", SYMLINK+="fitbit", MODE="0666"
```

```
$ cat /etc/udev/rules.d/60-vboxdrv.rules
```

```
KERNEL=="vboxdrv", NAME="vboxdrv", OWNER="root", GROUP="vboxusers", MODE="0660"
KERNEL=="vboxdrvu", NAME="vboxdrvu", OWNER="root", GROUP="root", MODE="0666"
KERNEL=="vboxnetctl", NAME="vboxnetctl", OWNER="root", GROUP="vboxusers", MODE="0660"
SUBSYSTEM=="usb_device", ACTION=="add", RUN+="/usr/lib/virtualbox/VBoxCreateUSBNode.sh $major $minor
↪  $attr{bDeviceClass}"
SUBSYSTEM=="usb", ACTION=="add", ENV{DEVTYPE}=="usb_device", RUN+="/usr/lib/virtualbox/VBoxCreateUSBNode.sh $major
↪  $minor $attr{bDeviceClass}"
SUBSYSTEM=="usb_device", ACTION=="remove", RUN+="/usr/lib/virtualbox/VBoxCreateUSBNode.sh --remove $major $minor"
SUBSYSTEM=="usb", ACTION=="remove", ENV{DEVTYPE}=="usb_device", RUN+="/usr/lib/virtualbox/VBoxCreateUSBNode.sh
↪  --remove $major $minor"
```


© Copyright the Linux Foundation 2023. All rights reserved.

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"
```

1. /etc/udev/rules.d
2. /run/udev/rules.d
3. /usr/lib/udev/rules.d

© Copyright the Linux Foundation 2023. All rights reserved.
THE LINUX FOUNDATION | Training & Certification

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"
```

for how these rules are applied in real-time by running:

```
$ sudo udevadm monitor
```


© Copyright the Linux Foundation 2023. All rights reserved.

Video Demonstration Resources

using_udev_demo.mp4

1.
2.

3. /dev
4.
5. /dev

1. /etc/udev/rules.d/75-myusb.rules

```
$ cat /etc/udev/rules.d/75-myusb.rules
  SUBSYSTEM=="usb", SYMLINK+="myusb"
```

2.
3. `$ ls -lF /dev | grep myusb`
4.

```
$ umount /media/whatever
      /media/whatever
```

5. `$ ls -lF /dev | grep myusb`

© Copyright the Linux Foundation 2023. All rights reserved.

© Copyright the Linux Foundation 2023. All rights reserved.

27.1 IP Addresses
27.2 IPv4 Address Types
27.3 IPv6 Address Types
27.4 IP Address Classes
27.5 Netmasks
27.6 Hostnames
27.7 NTP
27.8 Labs

Learning Objectives

By the end of this session, you should be able to:

- Differentiate between different types of IPv4 and IPv6 addresses.
- Understand IP Address Classes.
- Understand the role of netmasks.
- Get, set, and change the system hostname.
- Use **NTP** to set, synchronize, and correct the time.

IP Addresses

- IP addresses are used to uniquely identify nodes across the internet.
- IP addresses are assigned through **RIR**s
- 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

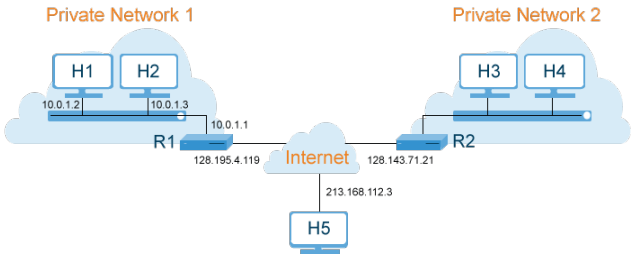


Figure 27.1: **IP Addresses**

© Copyright the Linux Foundation 2023. All rights reserved.

IPv4 Address Types

- Unicast
- Network
- Broadcast
- Multicast

© Copyright the Linux Foundation 2023. All rights reserved.

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`

- `127.x.x.x`
- `0.0.0.0`
- `255.255.255.255`
-
  `10.0.0.0 - 10.255.255.255`
  `172.16.0.0 - 172.31.255.255`
  `192.168.0.0 - 192.168.255.255`

https://en.wikipedia.org/wiki/Reserved_IP_addresses

© Copyright the Linux Foundation 2023. All rights reserved.

IPv6 Address Types

- Unicast
- Multicast
- Anycast
- IPv4-mapped

-
  - –
  - –
  - –
-
-
-

© Copyright the Linux Foundation 2023. All rights reserved.

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

© Copyright the Linux Foundation 2023. All rights reserved.

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

-
-
-
-
-

```
ex.  172.16.2.17  ip address
    &255.255.0.0  netmask
   ----------------
     172.16.0.0   network address
```


© Copyright the Linux Foundation 2023. All rights reserved.
THE LINUX FOUNDATION | Training & Certification

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
```

etc

/etc/sysconfig/network /etc/hostname /etc/HOSTNAME

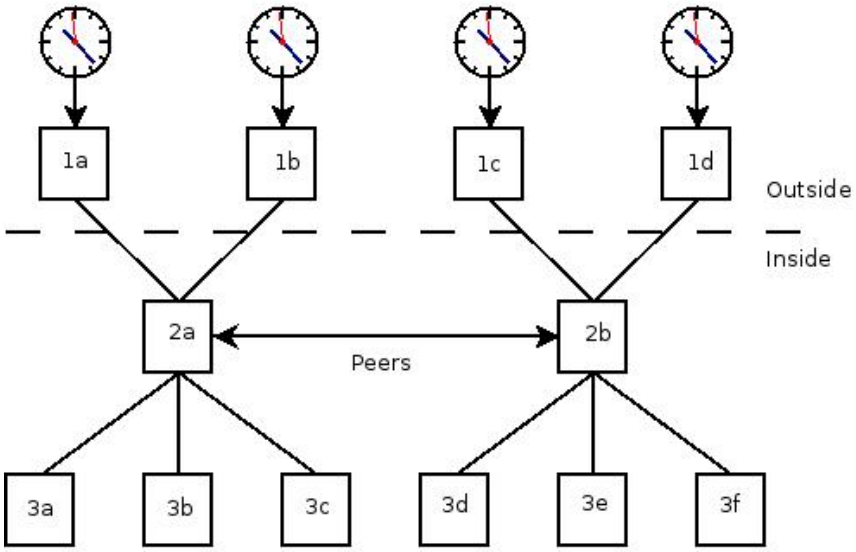
```
$ hostnamectl
   Static hostname: c8
         Icon name: computer-desktop
           Chassis: desktop
        Machine ID: ce0c82382a8a4c80bbd6931a917a2f1c
           Boot ID: 94207b3fbd9b4891b9a94e21762a47cb
  Operating System: Red Hat Enterprise Linux 8.2
  ↪  (Ootpa) \
           dracut-049-70.git20200228.el8 (Initramfs)
            Kernel: Linux 5.11.6
      Architecture: x86-64
```

and to see a usage message:

```
$ hostnamectl --help
hostnamectl [OPTIONS...] COMMAND ...
Query or change system hostname.
  -h --help              Show this help
     --version           Show package version
     --no-ask-password   Do not prompt for password
  -H --host=[USER@]HOST  Operate on remote host
  -M --machine=CONTAINER Operate on local container
     --transient         Only set transient hostname
     --static            Only set static hostname
     --pretty            Only set pretty hostname
Commands:
  status                 Show current hostname
  ↪  settings
  set-hostname NAME      Set system hostname
  set-icon-name NAME     Set icon name for host
  set-chassis NAME       Set chassis type for host
  set-deployment NAME    Set deployment environment
  ↪  for host
  set-location NAME      Set location for host
  set-location NAME      Set location for host
```


© Copyright the Linux Foundation 2023. All rights reserved.

Network Time Protocol



Key: 1 = stratum-1, 2 = stratum-2, 3 = stratum-3

Figure 27.2: **Network Time Protocol**

-
-
-
-
-
-
-

© Copyright the Linux Foundation 2023. All rights reserved.

NTP Applications

There are several **NTP** applications, some of the most common are:

- **ntp** ,http://www.ntp.org, the default.
- **chrony**, http://chrony.tuxfamily.org, Designed to work in a wide range of environments including intermittent network connections and virtual machines.
- **systemd-timesyncd**, a ntp client only included in the **systemd** package.

- `/etc/ntp.conf`
- `/etc/chrony.conf`
- `/etc/systemd/timesyncd.conf`

© Copyright the Linux Foundation 2023. All rights reserved.

Configuring the ntpd Client

- Choose an NTP source as a server or peer.
- Start NTP daemon
- Verify time is synchronizing

© Copyright the Linux Foundation 2023. All rights reserved.

http://www.pool.ntp.org/en/

/etc/ntp.conf

/etc/ntp.conf

```
driftfile /var/lib/ntp/ntp.drift

pool 0.pool.ntp.org
pool 1.pool.ntp.org
pool 2.pool.ntp.org
pool 3.pool.ntp.org
```

```
# timedatectl
```

```
               Local time: Sun 2023-03-05 16:13:50 UTC
           Universal time: Sun 2023-03-05 16:13:50 UTC
                 RTC time: Sun 2023-03-05 16:13:50
                Time zone: Etc/UTC (UTC, +0000)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```


© Copyright the Linux Foundation 2023. All rights reserved.

Configuring the ntpd Server

The configuration elements for the **ntp server** are contained in the `/etc/ntp.conf` file. Here are some of the **ntp server** relevant items:

- Declare the local machine to be a time reference
  - The **server** and **fudge** directives
- Regulate who can query the time server with **ntpq** and **ntpdc** commands (see CVE-2013-5211)
  - The **restrict** directives
- Declaration of which systems are **ntp peers**
  - The **peers** are listed in the `/etc/ntp.conf` file.
- Start NTP daemon

© Copyright the Linux Foundation 2023. All rights reserved.

/etc/ntp.conf

/etc/ntp.conf: access control

```
# Default policy prevents queries
restrict default nopeer nomodify notrap noquery
# Allow queries from a particular subnet
restrict 123.123.x.0 mask 255.255.255.0 nopeer nomodify notrap
# Allow queries from a particular host
restrict 131.243.1.42 nopeer nomodify notrap noquery
# Unrestrict localhost
restrict 127.0.0.1
```

/etc/ntp.conf

```
peer 128.100.49.12
peer 192.168.0.1
```

/etc/ntp.conf

```
server 127.127.1.0
fudge  127.127.1.0 stratum 10
```


© Copyright the Linux Foundation 2023. All rights reserved.
THE LINUX FOUNDATION Training & Certification

1.

```
# yum install ntp
# zypper install ntp
# apt install ntp
```

2.

/etc/ntp.conf

/etc/ntp.conf

```
server 0.pool.ntp.org
server 1.pool.ntp.org
server 2.pool.ntp.org
server 3.pool.ntp.org
```

3.

```
# systemctl restart ntpd
# systemctl restart ntp
```

4.

```
$ ntpq -p
```

Please Note

You may have to wait a few minutes for the timeservers to sync prior to getting any output.

© Copyright the Linux Foundation 2023. All rights reserved.

© Copyright the Linux Foundation 2023. All rights reserved.

28.1 Network Devices
28.2 Predictable Network Interface Device Names
28.3 Network Configuration Files
28.4 Network Manager
28.5 Routing
28.6 Virtual Network Interfaces
28.7 DNS and Name Resolution
28.8 Network Troubleshooting
28.9 Network Diagnostics
28.10 Labs

Learning Objectives

By the end of this session, you should be able to:

- Identify network devices and understand how the operating system names them and binds them to specific duties.
- Use the **ip** utility to display and control devices, routing, policy-based routing, and tunneling.
- Understand the Predictable Network Interface Device Names (**PNIDN**) scheme.
- Know the main network configuration files in the `/etc` directory.
- Use **Network Manager** (**nmtui** and **nmcli**) to configure network interfaces in a distribution-independent manner.
- Know how to set default routes and other static routes.
- Understand virtual network interfaces
- Understand DNS and Name Resolution
- Troubleshoot networks and run valuable diagnostic utilities.

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

`/dev`

© Copyright the Linux Foundation 2023. All rights reserved.

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`

© Copyright the Linux Foundation 2023. All rights reserved.

```
$ 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
```

```
$ lscpi | 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)
```

```
$ 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)
```


© Copyright the Linux Foundation 2023. All rights reserved.

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

© Copyright the Linux Foundation 2023. All rights reserved.

Network Manager Config Files

- **Network Manager** can use the traditional network files from several distributions. The networking profiles are supported by **plugins**.
- The **key-value** is the preferred file format.
- There are several optional `plugins` for traditional configuration compatibility.
  - **ifupdown** for `/etc/network/interfaces`
  - **ifcfg-rh** for `/etc/sysconfig/network-scripts`
  - **ifcfg-suse** is for simple compatibility for SUSE and openSUSE
  - **key-file** is a generic replacement for system specific configuration files.

For additional information see:
https://wiki.gnome.org/Projects/NetworkManager/SystemSettings/

`/etc/NetworkManager/NetworkManager.conf`

https://developer-old.gnome.org/NetworkManager/stable/NetworkManager.conf.html

© Copyright the Linux Foundation 2023. All rights reserved.

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

`/etc`

© Copyright the Linux Foundation 2023. All rights reserved.

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 **m**anager **t**ext **u**ser **i**nterface is a rather simple navigable command line interface using the **ncurses** library interface

- **nmcli**

  network **m**anager **c**ommand **l**ine **i**nterface is a lower level method which the higher level interfaces can use. It is useful for scripts

© Copyright the Linux Foundation 2023. All rights reserved.

nmtui

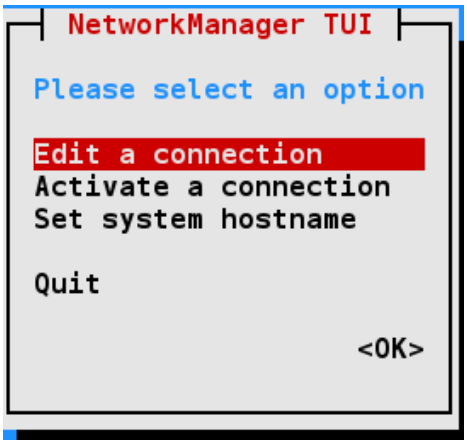


Figure 28.1: **nmtui Main Screen**

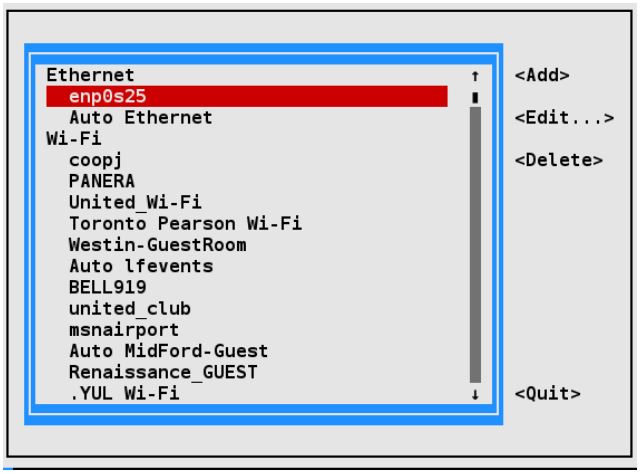


Figure 28.2: **nmtui Edit Screen**

© Copyright the Linux Foundation 2023. All rights reserved.
THE LINUX FOUNDATION | Training & Certification

nmtui Wireless Configuration

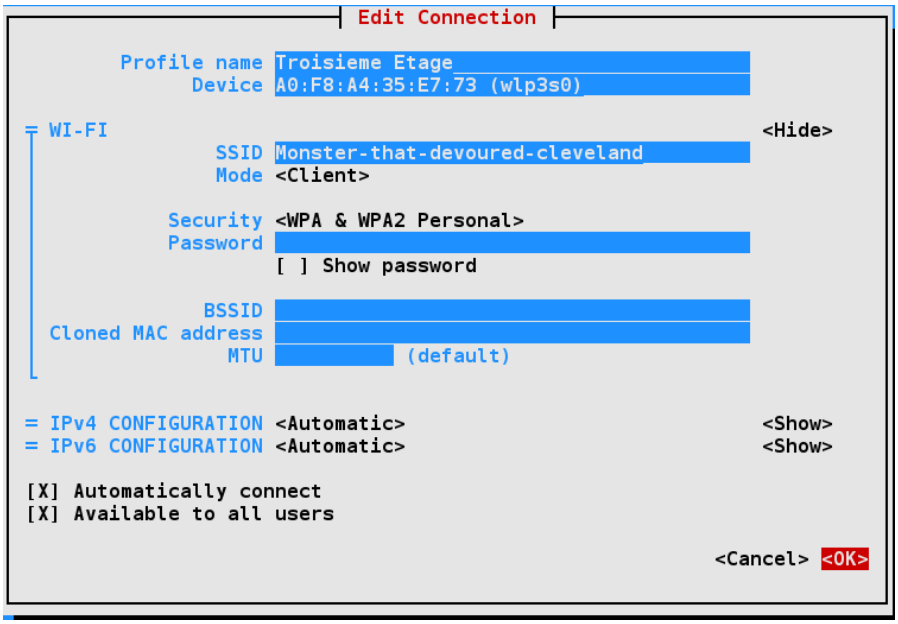


Figure 28.3: **nmtui Wireless Configuration**

© Copyright the Linux Foundation 2023. All rights reserved.

nmcli

- Command line interface to **Network Manager**
- Also has interactive mode
- Also `man nmcli-examples`
- Will try some examples in an exercise

Want More?

See https://fedoraproject.org/wiki/Networking/CLI

```
$ nmcli --help
```

```
Usage: nmcli [OPTIONS] OBJECT { COMMAND | help }
OPTIONS
  -a, --ask                                ask for missing parameters
  -c, --colors auto|yes|no                 whether to use colors in output
  -e, --escape yes|no                      escape columns separators in values
  -f, --fields <field,...>|all|common      specify fields to output
  -g, --get-values <field,...>|all|common  shortcut for -m tabular -t -f
  -h, --help                               print this help
  -m, --mode tabular|multiline             output mode
  -o, --overview                           overview mode
  -p, --pretty                             pretty output
  -s, --show-secrets                       allow displaying passwords
  -t, --terse                              terse output
  -v, --version                            show program version
  -w, --wait <seconds>                     set timeout waiting for finishing operations

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
```


© Copyright the Linux Foundation 2023. All rights reserved.

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

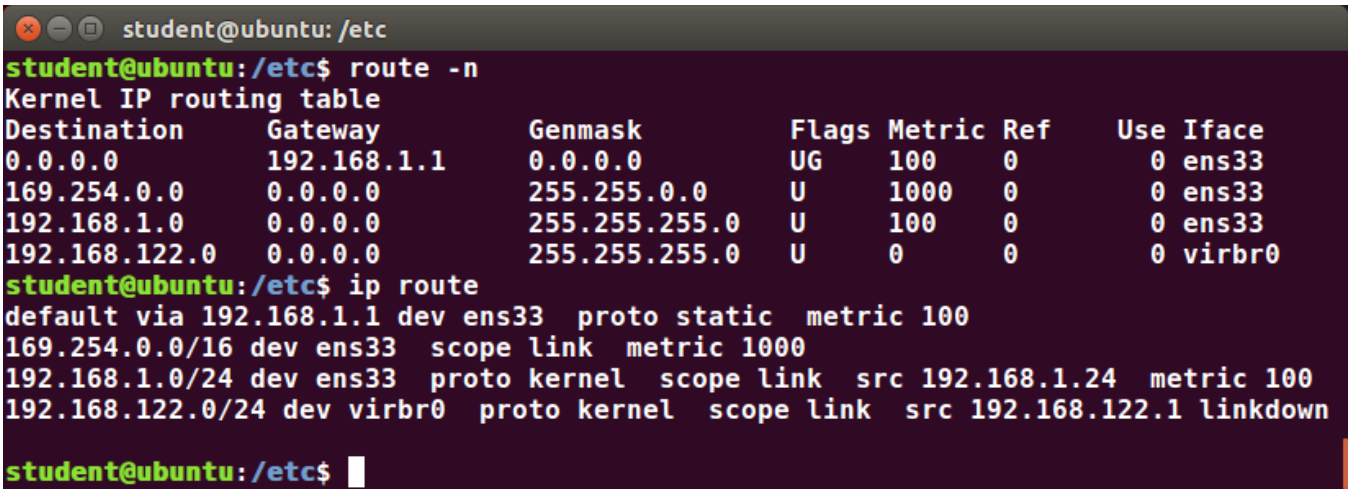


Figure 28.4: **Using route and ip route**

© Copyright the Linux Foundation 2023. All rights reserved.

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
```

```
$ sudo ip route add default via 192.168.1.10 dev enp2s0
$ ip route
```

```
default via 192.168.1.10 dev enp2s0
default via 192.168.1.1 dev enp2s0 metric 1024
192.168.1.0/24 dev enp2s0 proto kernel scope link src 192.168.1.100
```

```
$ sudo ip route add default via 192.168.1.1 dev enp2s0
```


© Copyright the Linux Foundation 2023. All rights reserved.

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)

```
$ sudo ip route add 10.5.0.0/16 via 192.168.1.100
```

/etc/sysconfig/network-scripts/route-ethX

```
$ cat /etc/sysconfig/network-scripts/route-eth0
  10.5.0.0/16 via 172.17.9.1
```

/etc/network/interfaces

```
iface eth1 inet dhcp
  post-up ip route add 10.1.2.51 dev eth1
  post-up ip route add 10.1.2.52 dev eth1
```

/etc/sysconfig/network/ifroute-eth0

```
# 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
```


© Copyright the Linux Foundation 2023. All rights reserved.

Teaming and Bonding Network Interfaces

- Teaming and Bonding interfaces allow dynamic network interface configuration to provide higher throughput and additional robustness to the network.
  - **Bonding**
    Uses a kernel module to provide aggregation of network links to provide fail-over, redundancy and performance enhancements
  - **Teaming**
    Is similar to **bonding** except **teaming** uses a userspace service named **teamed**.

Teaming is considered deprecated!

- The **Teaming** configuration method has been listed as **deprecated** and is scheduled to be removed in the future.
- To facilitate a smooth conversion from **Teaming** to **Bonding**, the **team2bond** utility is available to convert configuration files.

© Copyright the Linux Foundation 2023. All rights reserved.

Bonding Network Interfaces

There are several methods for configuring **Bonding** interfaces:

- **sysfs**
  Direct changes to the `/sys` pseudo filesystem. Changes are not saved.
- **iproute2**
  Supports **bonding** via the `ip` command, see `man 8 ip-link` for details. Changes are not saved.
- **NetworkManager**
  All **NetworkManager** user interfaces support **bonding**. Changes are saved.
  The **NetworkManager** configuration files can also be edited with your favorite text editor.

Want More?

See https://www.kernel.org/doc/Documentation/networking/bonding.txt for more information on the `/sys` interface to bonding

© Copyright the Linux Foundation 2023. All rights reserved.

Creating Bonding Network Interface with nmcli

Creating a configuration for a bonding interface with **nmcli** involves a few steps. Minimal steps are:

- **Identify adapters**: Use `nmcli device status` .
- **Create bonding device**: Use `nmcli connection add`.
- **Attach interface to the bond**: Create a connection from the adapter to the **bond** device using `nmcli connection add`.
- **Set the bond adapter online**: Issue a `nmcli connection up` command for the **bond adapter**.
- **reboot**: Although the reboot is not absolutely required, it is strongly recommended. A reboot will clean up any configuration fragments left around.

© Copyright the Linux Foundation 2023. All rights reserved.

On the command line

```
nmcli connection add type bond \
     con-name bond0 \
     ifname bond0 \
     bond.options "mode=active-backup,miimon=1000"
nmcli connection add type ethernet slave-type bond \
     con-name bond0-port1 \
     ifname enp0s20u1u1 \
     master bond0
nmcli connection add type ethernet slave-type bond \
     con-name bond0-port2 \
     ifname enp0s20u1u2 \
     master bond0
nmcli connection up bond0
nmcli device status
reboot
nmcli device status
```

Want More?

See https://wiki.gnome.org/Projects/NetworkManager/SystemSettings/ for more on **nmcli** and https://www.kernel.org/doc/Documentation/networking/bonding.txt for more on bonding and bonding options.

© Copyright the Linux Foundation 2023. All rights reserved.

/etc/hosts

- Used for local or static hostname resolution
- Useful for small isolated networks
- Usually checked before DNS

/etc/hosts

/etc/hosts

/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
```

/etc/nsswitch.conf

/etc /etc/hosts.deny /etc/hosts.allow

/etc/host.conf

© Copyright the Linux Foundation 2023. All rights reserved.
THE LINUX FOUNDATION Training & Certification

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
```

`/etc/hosts`

-
-
-

© Copyright the Linux Foundation 2023. All rights reserved.

Using Dig

- Gives the most correct answer among shipped tools
- Default output is often too verbose for simple tasks or scripting
- Can view entire resolution chain
- Can view and verify DNSSEC operation
- Some useful options:

  +trace Check the entire path, from the root servers (by default)

  -t [record type] specify the record (including `A`, `TXT`, `CNAME`, `MX`)

© Copyright the Linux Foundation 2023. All rights reserved.

```
$ dig linuxfoundation.org
```

```
; <<>> DiG 9.16.27-Debian <<>> linuxfoundation.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 54765
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;linuxfoundation.org.                IN        A

;; ANSWER SECTION:
linuxfoundation.org.        524        IN        A        3.13.31.214

;; Query time: 3 msec
;; SERVER: 10.139.1.1#53(10.139.1.1)
;; WHEN: Sun May 07 13:40:00 EDT 2023
;; MSG SIZE  rcvd: 64
```

```
$ dig -x 1.1.1.1 +short
```

```
one.one.one.one.
```

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** (**D**ynamic **H**ost **C**onfiguration **P**rotocol).

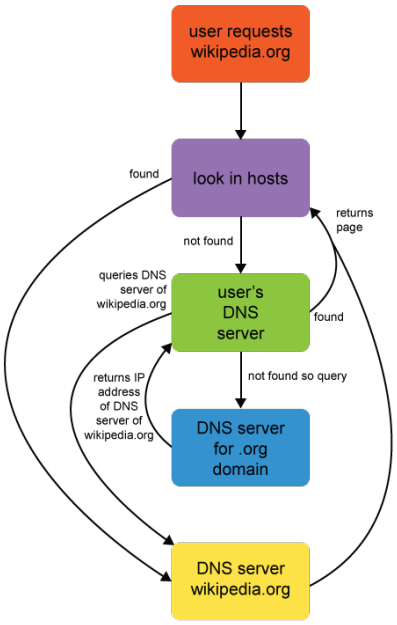


Figure 28.5: **DNS**

© Copyright the Linux Foundation 2023. All rights reserved.

/etc/hosts

/etc/resolv.conf

```
nameserver 192.168.1.1
nameserver 8.8.8.8
```

/etc/resolv.conf

```
# Generated by NetworkManager
```

```
192.168.1.1
```

which was generated by **NetworkManager** invoking **DHCP** on the primary network interface.

© Copyright the Linux Foundation 2023. All rights reserved.

28.8 Network Troubleshooting

Networking Problem Troubleshooting

- IP configuration
- Network Driver
- Connectivity
- Default gateway and routing configuration
- Hostname resolution

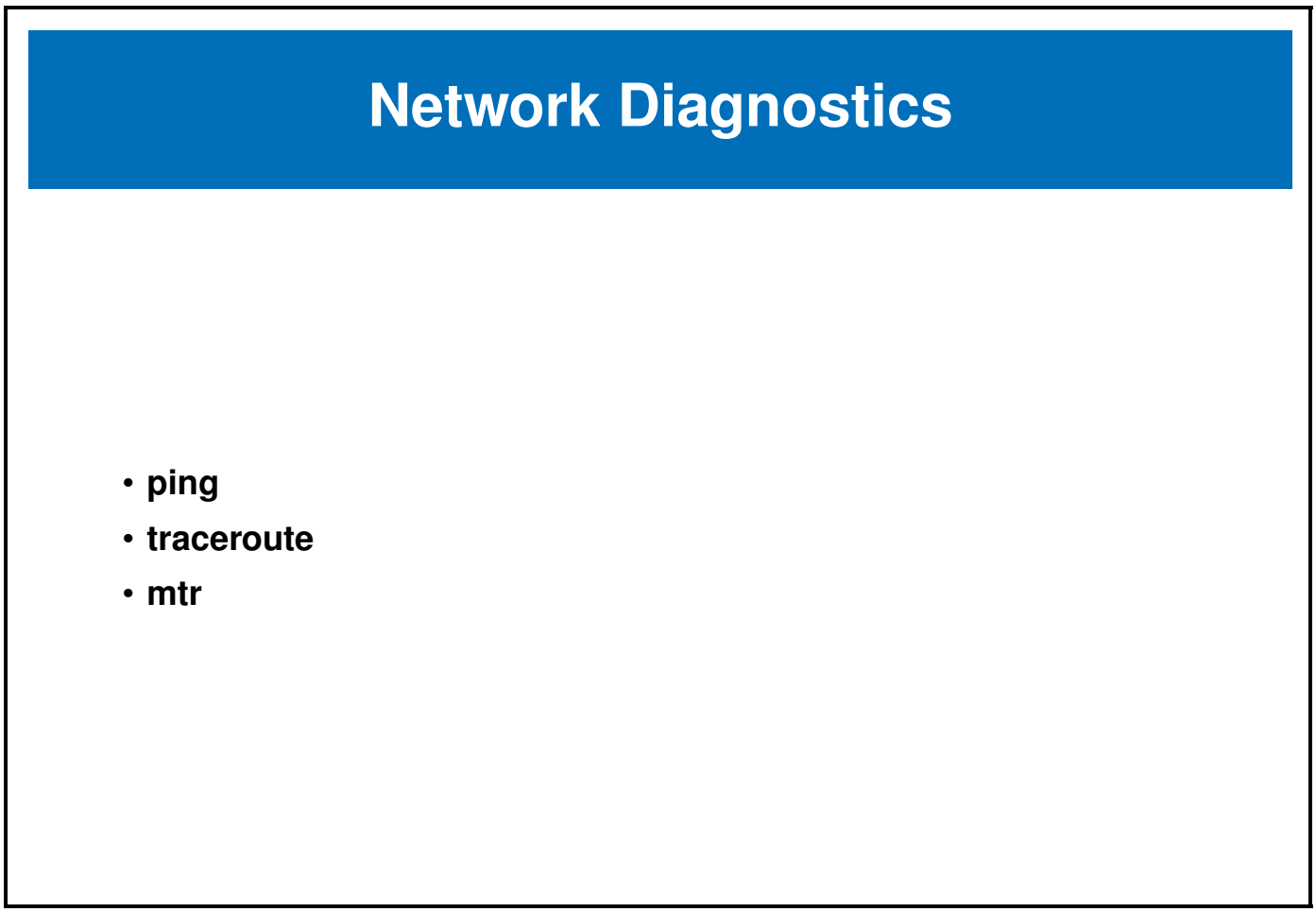
Network problems can be caused either by software or hardware, and can be as simple as is the device driver loaded, or is the network cable connected. If the network is up and running but performance is terrible, it really falls under the banner of performance tuning, not troubleshooting. The problems 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.

© Copyright the Linux Foundation 2023. All rights reserved.

28.9 Network Diagnostics



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**.

Note that some **Linux** distributions, such as **RHEL**, may require root privilege (as with **sudo**) in order to run these utilities.

© Copyright the Linux Foundation 2023. All rights reserved.

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 28.6: **ping**

$ **ping**

FIND THE ROUND TRIP TIME...

-c **10**

FOR A TOTAL OF **10** PACKETS ...

linuxfoundation.org

TO THE IP ADDRESS (RANDOM IF MORE THAN ONE) RETURNED BY THE CONFIGURED RESOLVER TO **linuxfoundation.org** ...

FINALLY, SUMMARIZE THE RESULTS (DEFAULT BEHAVIOR)

© Copyright the Linux Foundation 2023. All rights reserved.

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 28.7: **traceroute**

$ **traceroute**

FIND THE NETWORK PATH (BY DEFAULT USING A UDP PORT)...

google.com

TO THE IP ADDRESS (RANDOM IF MORE THAN ONE) RETURNED BY THE CONFIGURED RESOLVER TO **google.com**

Hidden Complexity

The hostname lookup for **google.com** will return a different set of IP addresses dependent on your network location using a variety of **DNS** and internet routing techniques.

© Copyright the Linux Foundation 2023. All rights reserved.

mtr Example

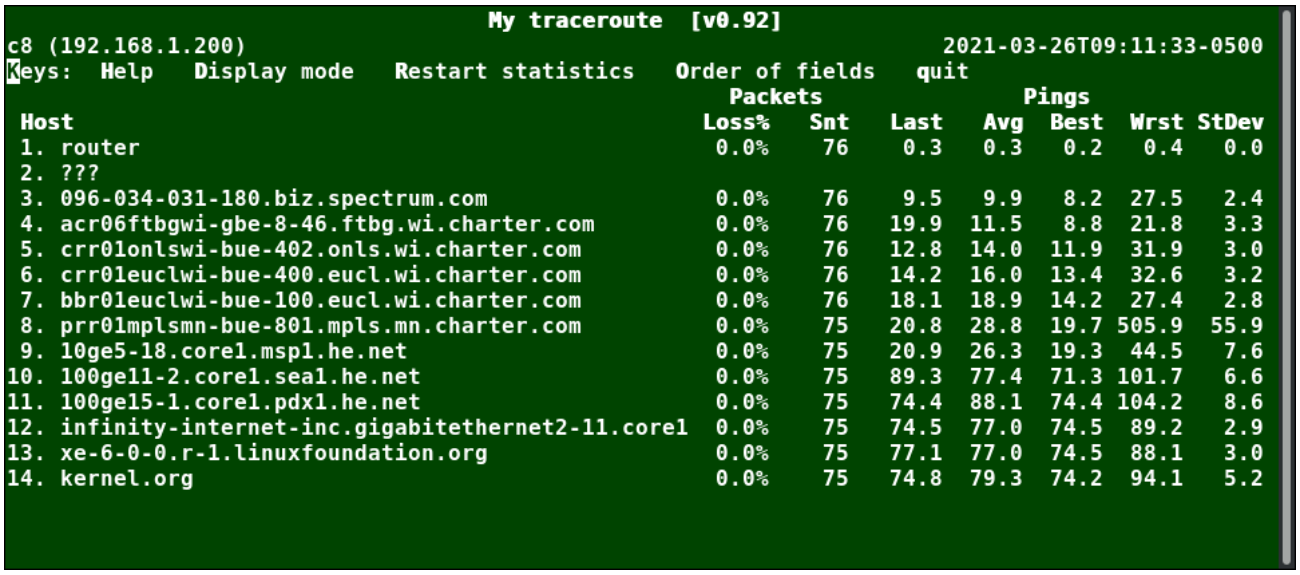


Figure 28.8: **mtr**

$ **mtr**

FIND THE NETWORK PATH (USING EITHER A UDP PORT OR ICMP)...

-t

USING THE TERMINAL AND NOT A GUI...

kernel.org

TO THE IP ADDRESS (RANDOM IF MORE THAN ONE) RETURNED BY THE CONFIGURED RESOLVER TO **kernel.org**

© Copyright the Linux Foundation 2023. All rights reserved.

28.10 Labs

Exercise 28.1: 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 28.1

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
```

```
--2014-11-01 08:57:12--  http://winhelp2002.mvps.org/hosts.txt
Resolving winhelp2002.mvps.org (winhelp2002.mvps.org)... 216.155.126.40
Connecting to winhelp2002.mvps.org (winhelp2002.mvps.org)|216.155.126.40|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 514744 (503K) [text/plain]
Saving to: hosts.txt

100%[======================================>] 514,744      977KB/s   in 0.5s

2014-11-01 08:57:13 (977 KB/s) - hosts.txt saved [514744/514744]
```

```
$ sudo sh -c "cat hosts.txt >> /etc/hosts"
```

Exercise 28.2: Verify DNS information with dig

In this exercise we will verify that a forward `IPv4 A` record matches its reverse `PTR` record for a mail server with **dig**.

1. Make sure you have bind installed. If not:

On Debian, Ubuntu, or Linux Mint

apt install bind9-dnsutils

On Red Hat, Centos, or Fedora

dnf install bind-utils

2. Find the mail server IP address for **protonmail.com**.

© Copyright the Linux Foundation 2023. All rights reserved.

3. Verify that the reverse lookup for the IP address matches.

Solution 28.2

1. `$ dig +short -t mx protonmail.com`

```
10 mailsec.protonmail.ch.
5 mail.protonmail.ch.
```

`$ dig +short -t A mail.protonmail.ch`

```
185.70.42.128
185.205.70.128
176.119.200.128
```

2. `$ dig +short -x 185.205.70.128`

```
mail.protonmail.ch.
```

Exercise 28.3: 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`

```
NAME           UUID                                  TYPE            DEVICE
Auto Ethernet  1c46bf37-2e4c-460d-8b20-421540f7d0e2  802-3-ethernet  ens33
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`

```
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"`

```
Connection successfully activated (D-Bus active path:
↪ /org/freedesktop/NetworkManager/ActiveConnection/6)
```


© Copyright the Linux Foundation 2023. All rights reserved.

```
$ ping -c 3 172.16.2.140
PING 172.16.2.140 (172.16.2.140) 56(84) bytes of data.
64 bytes from 172.16.2.140: icmp_seq=1 ttl=64 time=0.038 ms
64 bytes from 172.16.2.140: icmp_seq=2 ttl=64 time=0.034 ms
64 bytes from 172.16.2.140: icmp_seq=3 ttl=64 time=0.032 ms

--- 172.16.2.140 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
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 28.4: 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 **ip route**:

```
$ ip route
```

```
default via 172.16.2.2 dev ens33  proto static  metric 100
169.254.0.0/16 dev ens33  scope link  metric 1000
172.16.2.0/24 dev ens33  proto kernel  scope link  src 172.16.2.135  metric 100
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:

```
$ ip route
```

```
default via 172.16.2.2 dev ens33  proto static  metric 100
169.254.0.0/16 dev ens33  scope link  metric 1000
172.16.2.0/24 dev ens33  proto kernel  scope link  src 172.16.2.135  metric 100
192.168.122.0/24 dev virbr0  proto kernel  scope link  src 192.168.122.1 linkdown
```

4. Reload the interface to have it take effect and show it has:

```
$ sudo nmcli conn up "Auto Ethernet"
```

```
Connection successfully activated (D-Bus active path:
↪  /org/freedesktop/NetworkManager/ActiveConnection/25)
```

```
$ ip route
```

```
default via 172.16.2.2 dev ens33  proto static  metric 100
169.254.0.0/16 dev ens33  scope link  metric 1000
172.16.2.0/24 dev ens33  proto kernel  scope link  src 172.16.2.135  metric 100
```


© Copyright the Linux Foundation 2023. All rights reserved.

```
192.168.122.0/24 dev virbr0  proto kernel  scope link  src 192.168.122.1 linkdown
192.168.100.0/24 via 172.16.2.1 dev ens33
```

5. Reboot and verify the route has taken effect (i.e., it is **persistent**: If so remove it:

```
$ ip route
```

```
default via 172.16.2.2 dev ens33  proto static  metric 100
169.254.0.0/16 dev ens33  scope link  metric 1000
172.16.2.0/24 dev ens33  proto kernel  scope link  src 172.16.2.135  metric 100
192.168.122.0/24 dev virbr0  proto kernel  scope link  src 192.168.122.1 linkdown
192.168.100.0/24 via 172.16.2.1 dev ens33
```

```
$ sudo nmcli conn mod "Auto Ethernet" -ipv4.routes "192.168.100.0/24 172.16.2.1"

$ sudo nmcli conn up "Auto Ethernet"
```

```
Connection successfully activated (D-Bus active path:
↪  /org/freedesktop/NetworkManager/ActiveConnection/3)
$ ip route
default via 172.16.2.2 dev ens33  proto static  metric 100
169.254.0.0/16 dev ens33  scope link  metric 1000
172.16.2.0/24 dev ens33  proto kernel  scope link  src 172.16.2.135  metric 100
192.168.122.0/24 dev virbr0  proto kernel  scope link  src 192.168.122.1 linkdown
```

6. Note you can set a route with **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
...
```

You can verify that a route established this way is not persistent.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 29

LDAP **

29.1 LDAP Authentication 446
29.2 Labs ** 450

Resource Requirements

This exercise requires an additional Virtual Machine be installed. It is a small **LDAP server** that is used for testing our client configurations. The **LDAP** appliance uses about 512MB of memory and 2GB of disk space. Please complete this exercise outside of class.

Learning Objectives

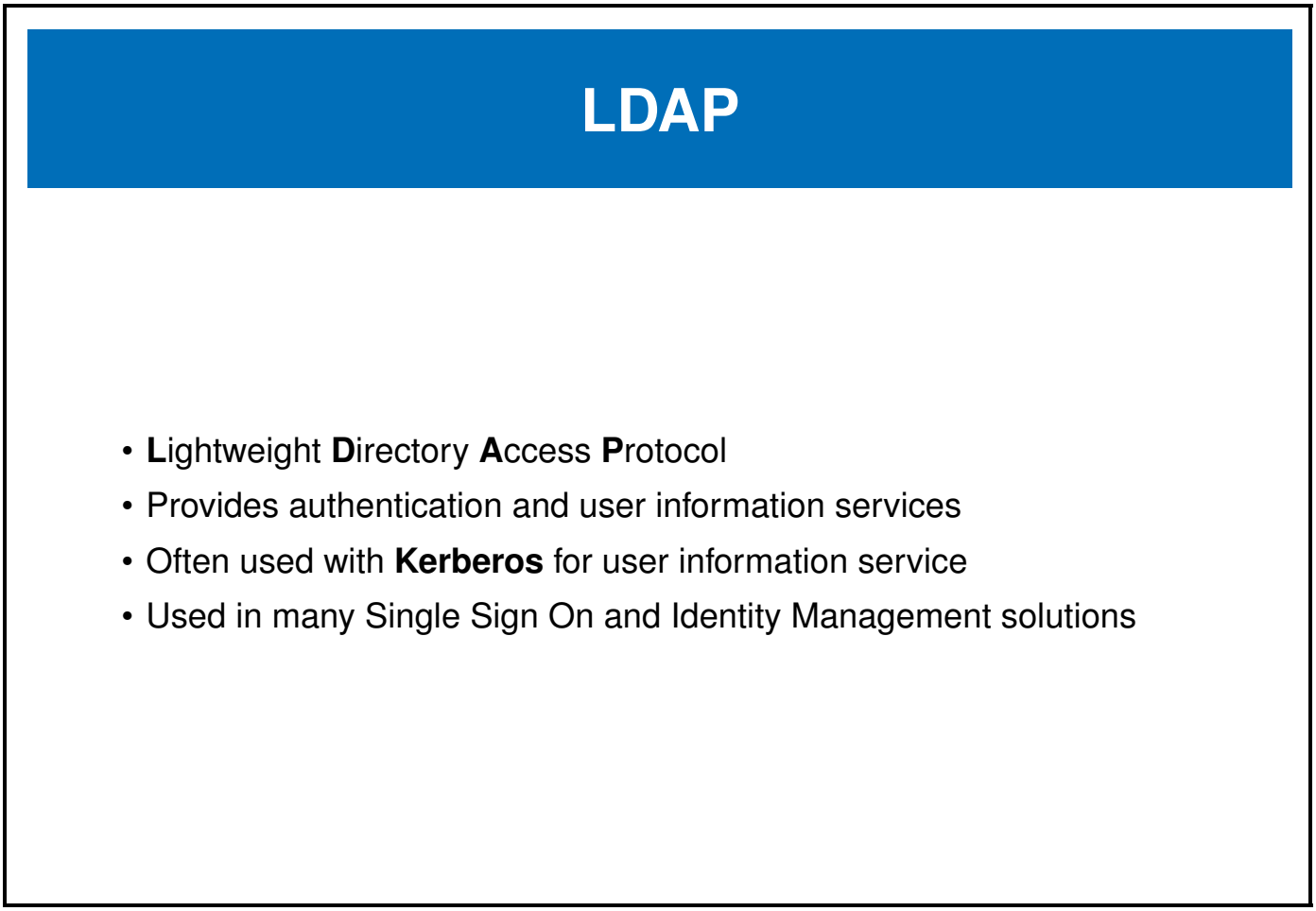
By the end of this session, you should be able to:

- Understand the purpose of **LDAP** Authentication.
- Know the important files in `/etc` that are involved in **LDAP**
- Know which utilities are available to work with these files.
- Work through an exercise with an **LDAP** server appliance.

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.

29.1 LDAP Authentication



LDAP is a derivative of the x.500 specification for providing directory services (think phone book) via the IP protocol. **LDAP** provides database-style functionality specifically engineered to be fast, computationally lightweight, and robust. The **LDAP** application protocol details transport, directory layout (organization), and methods of access to the data for operations such as read, write, delete, etc.

LDAP can provide both authentication **and** authorization data, however; it is primarily used for authorization alone, with **Kerberos** (trusted third party authentication) being a popular choice for authentication. Interestingly, **Microsoft**'s **Active Directory** service (at least for auth/auth) is little more than **LDAP** and **Kerberos** coupled together.

© Copyright the Linux Foundation 2023. All rights reserved.

LDAP Authentication

- Centralized authentication
- Multiple Domains
- Multi Platform
- multi-vendor
- Single sign on
- Uses **PAM** and **sssd**

The **LDAP** (**L**ightweight **D**irectory **A**ccess **P**rotocol) 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.

LDAP is **TLS** (**T**ransport **L**ayer **S**ecurity) and **SSL** **S**ecure **S**ockets **L**ayer aware. **TLS**/**SSL** will provide a secure encrypted connection.

The client configuration only needs to specify a minimum amount of information to connect to a **LDAP** server. Elements such as LDAP server IP Address, search base **DN** (domain name) and optionally **TLS** are required for authentication.

Additional user space programs are provided by the **Openldap-clients** package.

© Copyright the Linux Foundation 2023. All rights reserved.

LDAP Authentication Process

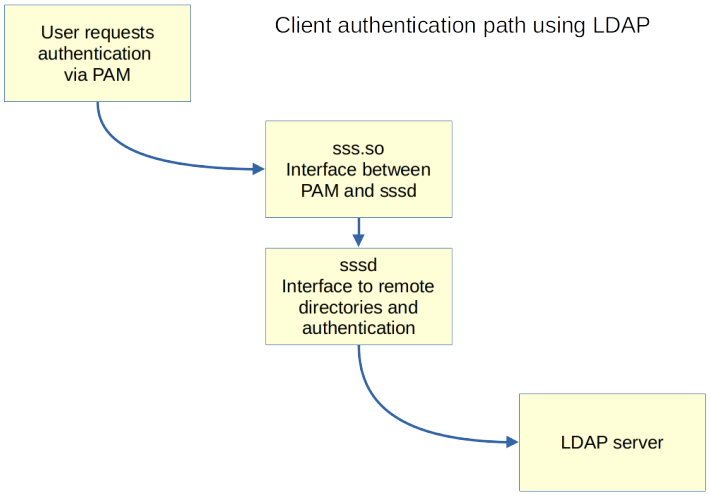


Figure 29.1: **LDAP Client Authentication Process**

The graphic above shows the information path used by **LDAP/SSSD**. Connecting to a **LDAP** server starts with **PAM**, the pluggable authentication module. **PAM** the is the default authentication mechanism for Linux, LDAP and SSSD are extension modules added to the default authentication configuration.

- **PAM** is configured to use **pam.sss.so module**
- **sss.so** and **sssd** use a combined **sssd** and **LDAP** configuration file.

/etc/sssd/conf.d/00-sssd.conf

```
[sssd]
config_file_version = 2
domains = example.com
services = nss, pam,autofs

[domain/example.com]
enumerate = true
id_provider = ldap
autofs_provider = ldap
auth_provider = ldap
chpass_provider = ldap
ldap_uri = ldap://192.168.122.154/
ldap_search_base = dc=example,dc=com
ldap_id_use_start_tls = true
cache_credentials = True
ldap_tls_reqcert =allow
```


© Copyright the Linux Foundation 2023. All rights reserved.

LDAP and sssd packages

- **sssd** System Security Services Daemon
- **sssd-ldap** configuration mechanism for sssd when used with LDAP
- **sssd-tools** various tools used with sssd
- **oddjob-mkhomedir** PAM module that creates home directory
- **ldap-utils** userspace utilities, contains tool to add and delete users from ldap, specific to **Ubuntu**
- **openldap-clients** same as ldap-utils only for **CentOS**

When you configure a client system for **LDAP** authentication, The following files are changed:

```
/etc/sssd/conf.d/00-sssd.conf
/etc/pam.d/common-session.conf
```

The **/etc/sssd/conf.d/00-ssd.conf** file has a 2 digit prefix to allow for sequencing if more than one configuration file is being used.

The **/etc/pam.d/** configuration files may be different depending on your distribution.

- **CentOS**-8 /etc/pam.d/system-auth
- **Ubuntu**-22.04 /etc/pam.d/common-session.conf

You can edit these files manually.

In previous distribution releases there were specific utilities used for configuration. The adoption of **sssd** and friends has greatly reduced the complexity of the client configuration and eliminated the need for specialized configuration tools. Text file configuration information may be configured with your favorite text editor.

© Copyright the Linux Foundation 2023. All rights reserved.

29.2 Labs **

Exercise 29.1: Centralized Authentication using LDAP and TLS - Install an LDAP server appliance

Resource Requirements

This exercise requires an additional Virtual Machine be installed. It is a small **LDAP server** that is used for testing our client configurations. The **LDAP** appliance uses about 512MB of memory and 2GB of disk space. Please complete this exercise outside of class.

This exercise will install a **LDAP** server appliance for centralized network authentication. The **turnkey openldap** appliance server was chosen for its availability on different distributions. Installing this appliance will create a new **VM**. The **appliance VM** is quite small, assigning only 512MiB of memory and using less than 1GB of disk space. The disk space is dynamically grow-able to 20GB but our usage remains quite low. There are many options to obtain the appliance and one option, the OVA image that is compatible with **VMWare** and **Virtual-Box** is available in the following directory. Another option is a **vmdk** image file that works with **VMWare Player** and **VMWare Workstation** is also available in the directory below.

https://training.linuxfoundation.org/cm/LFS207

Please see https://www.turnkeylinux.org/openldap for available options.

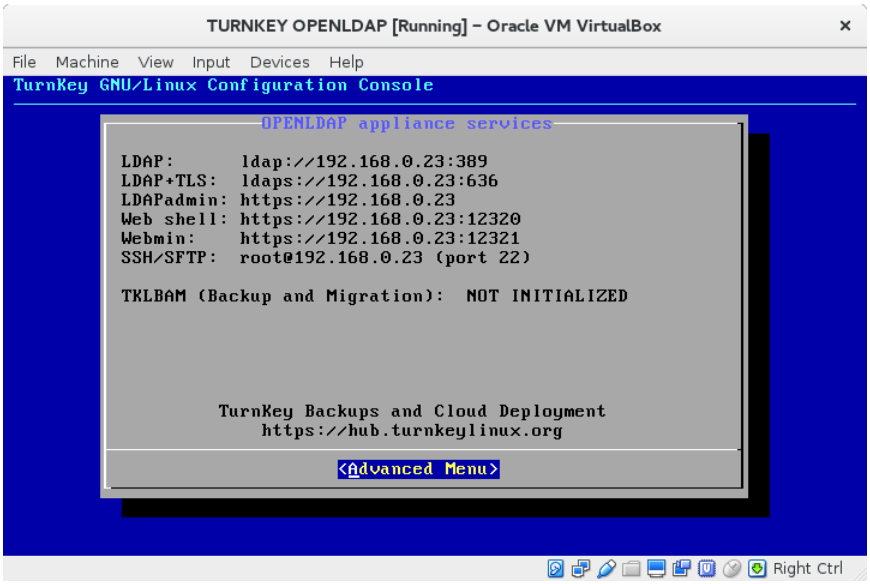


Figure 29.2: **Turnkey OpenLDAP Appliance Services**

1. Import the appliance
   - Verify the file `turnkey-openldap-14.1-jessie-amd64.ova` is available (version may be different).
   - Open the **Virtual-Box Manager**.
   - Select `File -> Import Appliance`
   - Enter location (or browse) of the **OpenLADP appliance**.
   - Select **Next** and continue as prompted.
2. Start the **TURNKEY OPENLDAP** virtual machine just created.
3. Configure the LDAP appliance on first boot with the following parameters:
   - `Root's password is "pa$$w0rd"`

© Copyright the Linux Foundation 2023. All rights reserved.

- Admin's password id "pa$$w0rd"
- The domain is "example.com"
- No "HUB","System Notification" or "Update" services

The **Turnkey OpenLDAP Appliance Summary** should be visible, this is the normal **running** display of the appliance.

Exercise 29.2: Centralized Authentication using LDAP and TLS - testing the LDAP server and adding users.

In this exercise we will verify the basic operation of the **LDAP** server. Specifically the ability of the server to respond to queries, and add a **POSIX** group and user. Once the user is added we can monitor the connection with **wireshark** and observe the plain text transfer of information to and from the **LDAP** server. Since it is a poor security policy to have clear text user information, we will encrypt the data stream.

Very Important

This lab is going to use clear text when communication with the **ldap** server. The use of encryption keys is out of scope for this course. Clear text is **NOT RECOMMENDED** in any sort of production environment. To accommodate our lab environment we will disable certificate verification. The ability to not verify the keys is handy in our lab but **DO NOT** use this technique on any production systems.

This exercise will be using the machine used in previous exercises to communicate with the **LDAP** server.

It is assumed that the `ready-for.sh` script has been run, the solutions and resource files loaded and extracted to a directory.

1. Verify or install openldap-clients packages, use the appropriate command.
   Ubuntu

   ```
   # apt install sssd sssd-ldap sssd-tools oddjob-mkhomedir ldap-utils
   ```

   CentOS

   ```
   # yum install sssd sssd-ldap sssd-tools oddjob-mkhomedir openldap-clients
   ```

2. Using the **ip address** of the **LDAP** server displayed in the **Turnkey OpenLDAP Appliance Summary** screen in the previous exercise, perform a simple **LDAP** search of the **example.com** domain. We should see information about the base records for the domain example.com

   ```
   # ldapsearch -x -H ldap://192.168.0.23 -b "dc=example,dc=com" -s sub"objectclass=*"
   ```

   ```
   # extended LDIF
   #
   # LDAPv3
   # base <dc=example,dc=com> with scope subtree
   # filter: (objectclass=*)
   # requesting: ALL
   #

   # example.com
   dn: dc=example,dc=com
   objectClass: top
   objectClass: dcObject
   objectClass: organization
   o: example.com
   dc: example

   # admin, example.com
   dn: cn=admin,dc=example,dc=com
   objectClass: simpleSecurityObject
   objectClass: organizationalRole
   ```


© Copyright the Linux Foundation 2023. All rights reserved.

```
cn: admin
description: LDAP administrator

# Groups, example.com
dn: ou=Groups,dc=example,dc=com
objectClass: organizationalUnit
objectClass: top
ou: Groups

# users, Groups, example.com
dn: cn=users,ou=Groups,dc=example,dc=com
cn: users
gidNumber: 100
objectClass: posixGroup
objectClass: top

# Users, example.com
dn: ou=Users,dc=example,dc=com
objectClass: organizationalUnit
objectClass: top
ou: Users

# Hosts, example.com
dn: ou=Hosts,dc=example,dc=com
objectClass: organizationalUnit
objectClass: top
ou: Hosts

# Idmaps, example.com
dn: ou=Idmaps,dc=example,dc=com
objectClass: organizationalUnit
objectClass: top
ou: Idmaps

# samba, example.com
dn: cn=samba,dc=example,dc=com
cn: samba
objectClass: simpleSecurityObject
objectClass: organizationalRole
description: SAMBA Access Account

# Aliases, example.com
dn: ou=Aliases,dc=example,dc=com
objectClass: organizationalUnit
objectClass: top
ou: Aliases

# nsspam, example.com
dn: cn=nsspam,dc=example,dc=com
cn: nsspam
objectClass: simpleSecurityObject
objectClass: organizationalRole
description: NSS/PAM Access Account

# search result
search: 2
result: 0 Success

# numResponses: 11
# numEntries: 10
```


© Copyright the Linux Foundation 2023. All rights reserved.

3. Add a new group record using the `groups.ldif` file included in the **solutions** directory.

```
# ldapadd -x -D "cn=admin,dc=example,dc=com" -W -H ldap://192.168.0.23 -f groups.ldif
```

```
 Enter LDAP Password:
adding new entry "cn=luser1,ou=Groups,dc=example,dc=com "
```

4. The **ldapsearch** command should now show a new group.

```
# ldapsearch -x -H ldap://192.168.0.23 -b "dc=example,dc=com" -s sub"objectclass=*"
```

```
 extended LDIF
#
# LDAPv3
# base <dc=example,dc=com> with scope subtree
# filter: (objectclass=*)
# requesting: ALL
#

# example.com
dn: dc=example,dc=com
objectClass: top
objectClass: dcObject
objectClass: organization
o: example.com
dc: example

# admin, example.com
dn: cn=admin,dc=example,dc=com
objectClass: simpleSecurityObject
objectClass: organizationalRole
cn: admin
description: LDAP administrator

# Groups, example.com
dn: ou=Groups,dc=example,dc=com
objectClass: organizationalUnit
objectClass: top
ou: Groups

# users, Groups, example.com
dn: cn=users,ou=Groups,dc=example,dc=com
cn: users
gidNumber: 100
objectClass: posixGroup
objectClass: top

# Users, example.com
dn: ou=Users,dc=example,dc=com
objectClass: organizationalUnit
objectClass: top
ou: Users

# Hosts, example.com
dn: ou=Hosts,dc=example,dc=com
objectClass: organizationalUnit
objectClass: top
ou: Hosts

# Idmaps, example.com
dn: ou=Idmaps,dc=example,dc=com
objectClass: organizationalUnit
```


© Copyright the Linux Foundation 2023. All rights reserved.

```
objectClass: top
ou: Idmaps

# samba, example.com
dn: cn=samba,dc=example,dc=com
cn: samba
objectClass: simpleSecurityObject
objectClass: organizationalRole
description: SAMBA Access Account

# Aliases, example.com
dn: ou=Aliases,dc=example,dc=com
objectClass: organizationalUnit
objectClass: top
ou: Aliases

# nsspam, example.com
dn: cn=nsspam,dc=example,dc=com
cn: nsspam
objectClass: simpleSecurityObject
objectClass: organizationalRole
description: NSS/PAM Access Account

# luser1, Groups, example.com
dn: cn=luser1,ou=Groups,dc=example,dc=com
cn: luser1
objectClass: posixGroup
objectClass: top
gidNumber: 999001
memberUid: luser1

# search result
search: 2
result: 0 Success

# numResponses: 12
# numEntries: 11
```

Notice in the output above the **numResponses** and **numEntries** have increased when we added the new group record.

5. Add a new user record using the `users.ldif` file included in the **solutions** directory.

```
# ldapadd -x -D "cn=admin,dc=example,dc=com" -W -H ldap://192.168.0.23 -f users.ldif
```

```
Enter LDAP Password:
adding new entry "cn=luser1,dc=example,dc=com"
```

6. The **ldapsearch** command should now show a new user.

```
# ldapsearch -x -H ldap://192.168.0.23 -b "dc=example,dc=com" -s sub"objectclass=*"
```

```
# extended LDIF
#
# LDAPv3
# base <dc=example,dc=com> with scope subtree
# filter: (objectclass=*)
# requesting: ALL
#

# example.com
dn: dc=example,dc=com
```


© Copyright the Linux Foundation 2023. All rights reserved.

```
objectClass: top
objectClass: dcObject
objectClass: organization
o: example.com
dc: example

# admin, example.com
dn: cn=admin,dc=example,dc=com
objectClass: simpleSecurityObject
objectClass: organizationalRole
cn: admin
description: LDAP administrator

# Groups, example.com
dn: ou=Groups,dc=example,dc=com
objectClass: organizationalUnit
objectClass: top
ou: Groups

# users, Groups, example.com
dn: cn=users,ou=Groups,dc=example,dc=com
cn: users
gidNumber: 100
objectClass: posixGroup
objectClass: top

# Users, example.com
dn: ou=Users,dc=example,dc=com
objectClass: organizationalUnit
objectClass: top
ou: Users

# Hosts, example.com
dn: ou=Hosts,dc=example,dc=com
objectClass: organizationalUnit
objectClass: top
ou: Hosts

# Idmaps, example.com
dn: ou=Idmaps,dc=example,dc=com
objectClass: organizationalUnit
objectClass: top
ou: Idmaps

# samba, example.com
dn: cn=samba,dc=example,dc=com
cn: samba
objectClass: simpleSecurityObject
objectClass: organizationalRole
description: SAMBA Access Account

# Aliases, example.com
dn: ou=Aliases,dc=example,dc=com
objectClass: organizationalUnit
objectClass: top
ou: Aliases

# nsspam, example.com
dn: cn=nsspam,dc=example,dc=com
cn: nsspam
objectClass: simpleSecurityObject
objectClass: organizationalRole
description: NSS/PAM Access Account
```


© Copyright the Linux Foundation 2023. All rights reserved.

```
# luser1, Groups, example.com
dn: cn=luser1,ou=Groups,dc=example,dc=com
cn: luser1
objectClass: posixGroup
objectClass: top
gidNumber: 999001
memberUid: luser1

# luser1, example.com
dn: cn=luser1,dc=example,dc=com
uid: luser1
cn: luser1
givenName: luser1
sn: linux
homeDirectory: /home/users/luser1
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: top
uidNumber: 999001
gidNumber: 999001

# search result
search: 2
result: 0 Success

# numResponses: 13
# numEntries: 12
```

7. Install or verify wireshark is installed.

```
# yum install wireshark-gnome
# wireshark
```

8. Capture an ldapsearch command with wireshark, notice the data is clear text.

   Wireshark has excellent filters, use the filter `"tcp and (tcp.port==389) or (tcp.port==636)"` to have **wireshark** only display **LDAP** communication.

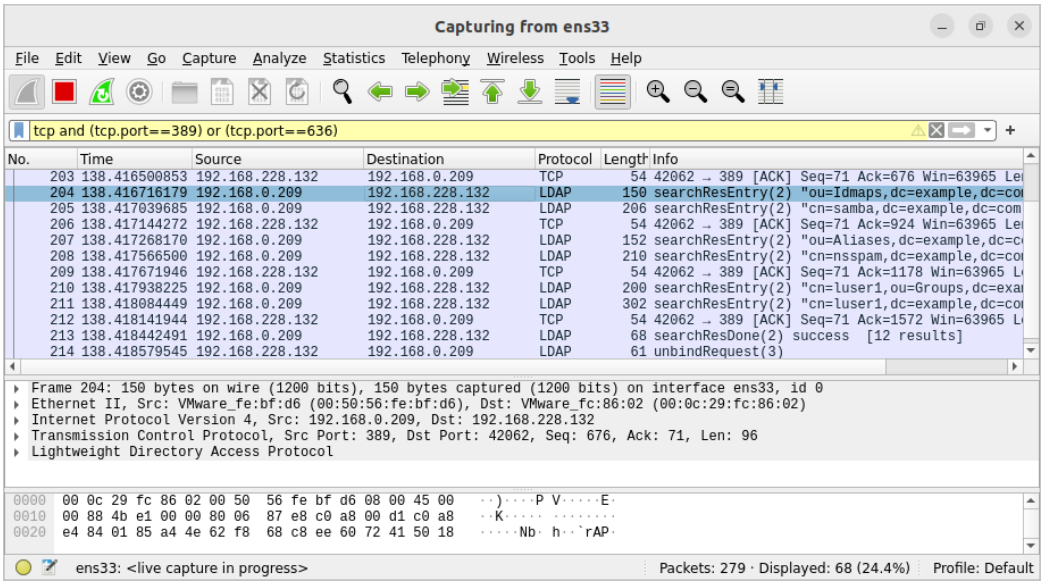


Figure 29.3: **Wireshark LDAP packet trace**

© Copyright the Linux Foundation 2023. All rights reserved.

Exercise 29.3: Centralized Authentication using LDAP for user authentication

In this exercise, we configure the client to use **LDAP** for centralized authentication.

Very Important

Starting with **CentOS-8** and **Ubuntu 20.04** the **System Security Services Daemon** or **sssd** is being used to manage access to remote directories and authentication. The use and configuration **sssd** is the same across the participating distributions.

1. Install required packages.

```
# apt install sssd sssd-ldap ldap-utils oddjob-mkhomedir
```

2. Create or update the **sssd** configuration to include the elements listed substituting your **ldap** server IP address in the **ldap_uri** field.

/etc/sssd/conf.d/00-sssd.conf

```
[sssd]
config_file_version = 2
domains = example.com
services = nss, pam,autofs

[domain/example.com]
enumerate = true
id_provider = ldap
autofs_provider = ldap
auth_provider = ldap
chpass_provider = ldap
ldap_uri = ldap://192.168.122.154/
ldap_search_base = dc=example,dc=com
ldap_id_use_start_tls = true
cache_credentials = True
ldap_tls_reqcert =allow
```

3. Verify and set the permissions for the **sssd.conf** file.

```
# chmod 600 /etc/sssd/conf.d/00-sssd.conf
# chown root.root /etc/sssd/conf.d/00-sssd.conf
```

4. Set up **oddjob** to automatically create home directories. Add **oddjob** to `/etc/pam.d/common-session` as in the following example:

/etc/pam.d/common-session.conf

```
session required        pam\_unix.so

session optional        pam\_oddjob\_mkhomedir.so

session optional        pam\_sss.so
```

5. Restart the services to pick up the changes.

```
# systemctl restart sssd oddjobd
# systemctl enable sssd oddjobd
```


© Copyright the Linux Foundation 2023. All rights reserved.

6. Test the **ldap** server.
7. Verify the user information is available for user **luser1**

```
# getent passwd luser1
```

The response should be:

```
luser1:*:999001:999001:luser1:/home/users/luser1:
```

8. Test the user authorization is functioning for user **luser1**. The password should be password. There may not be a home directory available.

```
# ssh luser1@localhost
```

9. Verify the data stream is clear text.
   - Open a **wireshark** session filtering on ports 389 and 636.
   - While **wireshark** is capturing packets, log on and off as user luser1 via ssh to localhost.
   - Observe the output in **wireshark**
10. Disable the certificate validation.

Very Important

Disabling the certificate validation **NOT RECOMMENDED in production systems.**

Edit the `/etc/sssd/sssd.conf` file with your favorite editor. In the `[domain/default]` section add the line:

```
ldap_tls_reqcert = never
```

Restart the `sssd` service.

```
# systemctl restart sssd
```

11. Test the user information is being supplied by **LDAP** for **luser1**

```
# getent passwd luser1
```

12. Test the user authorization is functioning for **luser1**.

```
# ssh luser1@localhost
```

13. Verify the data stream to and from the **LDAP** server is encrypted.
    - Open a **wireshark** session filtering on ports 389 and 636.
    - While **wireshark** is capturing packets, log on and off as user luser1 via ssh to localhost.
    - Observe the output in **wireshark**.
14. Restore the original configuration when completed.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 30

Firewalls

30.1 Firewalls 460
30.2 Interfaces 463
30.3 firewalld 465
30.4 Zones 468
30.5 Source Management 472
30.6 Service and Port Management 473
30.8 Labs 481

Learning Objectives

By the end of this session, you should be able to:

- Understand what firewalls are and why they are necessary.
- Know what tools are available both at the command line and using graphical interfaces.
- Discuss **firewalld** and the **firewall-cmd** programs.
- Know how to work with zones, sources, services and ports.

30.1 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.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

30.2 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 **GUI**s 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.

© Copyright the Linux Foundation 2023. All rights reserved.

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

© Copyright the Linux Foundation 2023. All rights reserved.

30.3 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

© Copyright the Linux Foundation 2023. All rights reserved.

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
```

```
Usage: firewall-cmd [OPTIONS...]
....
Status Options
  --state                Return and print firewalld state
  --reload               Reload firewall and keep state information
  --complete-reload      Reload firewall and loose state information
  --runtime-to-permanent Create permanent from runtime configuration
....
```

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.

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
```


© Copyright the Linux Foundation 2023. All rights reserved.

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
```

```
firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled)
   Active: active (running) since Tue 2015-04-28 12:00:59 CDT; 5min ago
 Main PID: 777 (firewalld)
...
```

```
$ 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.

© Copyright the Linux Foundation 2023. All rights reserved.

30.4 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.

© Copyright the Linux Foundation 2023. All rights reserved.

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: eno16777736
```

- List all available zones:

```
$ sudo firewall-cmd --get-zones
block dmz drop external home internal public trusted work
```

```
$ firewall-cmd --help
```

```
....
Zone Options
  --get-default-zone   Print default zone for connections and interfaces
  --set-default-zone=<zone>
                       Set default zone
  --get-active-zones   Print currently active zones
  --get-zones          Print predefined zones [P]
  --get-services       Print predefined services [P]
  --get-icmptypes      Print predefined icmptypes [P]
  --get-zone-of-interface=<interface>
                     Print name of the zone the interface is bound to [P]
  --get-zone-of-source=<source>[/<mask>]
                 Print name of the zone the source[/mask] is bound to [P]
  --list-all-zones  List everything added for or enabled in all zones [P]
  --new-zone=<zone>    Add a new zone [P only]
  --delete-zone=<zone> Delete an existing zone [P only]
  --zone=<zone>  Use this zone to set or query options, else default zone
                       Usable for options maked with [Z]
  --get-target         Get the zone target [P] [Z]
  --set-target=<target>
                       Set the zone target [P] [Z]
```


© Copyright the Linux Foundation 2023. All rights reserved.

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=eno1
success
```

- To do so permanently:

```
$ sudo firewall-cmd --permanent --zone=internal --change-interface=eno1
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
```


© Copyright the Linux Foundation 2023. All rights reserved.

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:

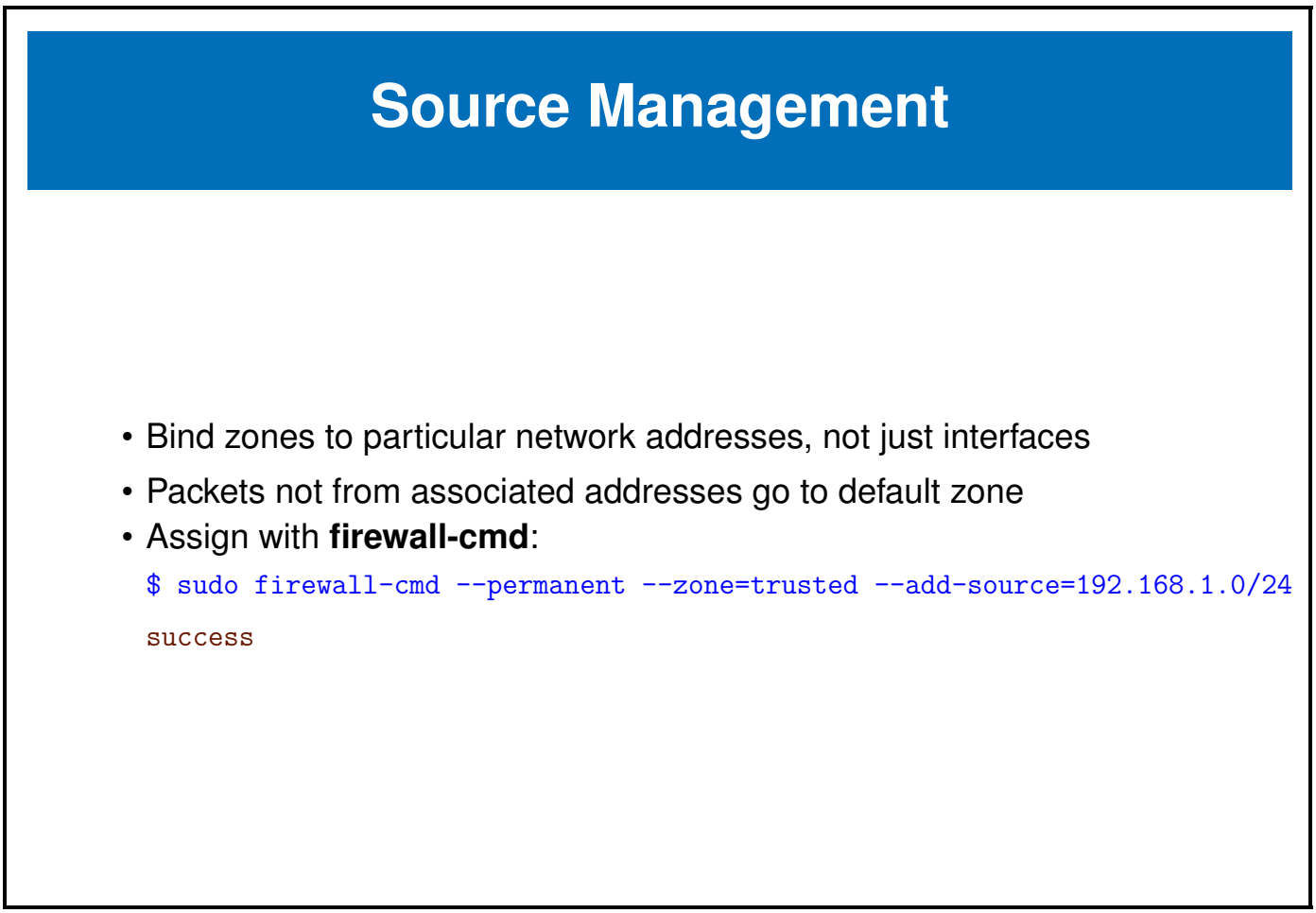
```
$ sudo firewall-cmd --zone=public --list-all
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: eno1
  sources:
  services: chromecast libvirt libvirt-tls nfs nfs3 rsyncd ssh
  ports:
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

Want More?

For a comprehesive reference on **firewall-cmd**, see:
https://firewalld.org/documentation/man-pages/firewall-cmd.html

© Copyright the Linux Foundation 2023. All rights reserved.

30.5 Source Management



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
  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.

© Copyright the Linux Foundation 2023. All rights reserved.

30.6 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
```


© Copyright the Linux Foundation 2023. All rights reserved.

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
```

```
RH-Satellite-6 amanda-client bacula bacula-client dhcp dhcpv6 dhcpv6-client dns ftp high-availability
↪ http https imaps ipp ipp-client ipsec kerberos kpasswd ldap ldaps libvirt libvirt-tls mdns mountd
↪ ms-wbt mysql nfs ntp openvpn pmcd pmproxy pmwebapi pmwebapis pop3s postgresql proxy-dhcp radius
↪ rpc-bind samba 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
```

```
dhcpv6-client ssh
```

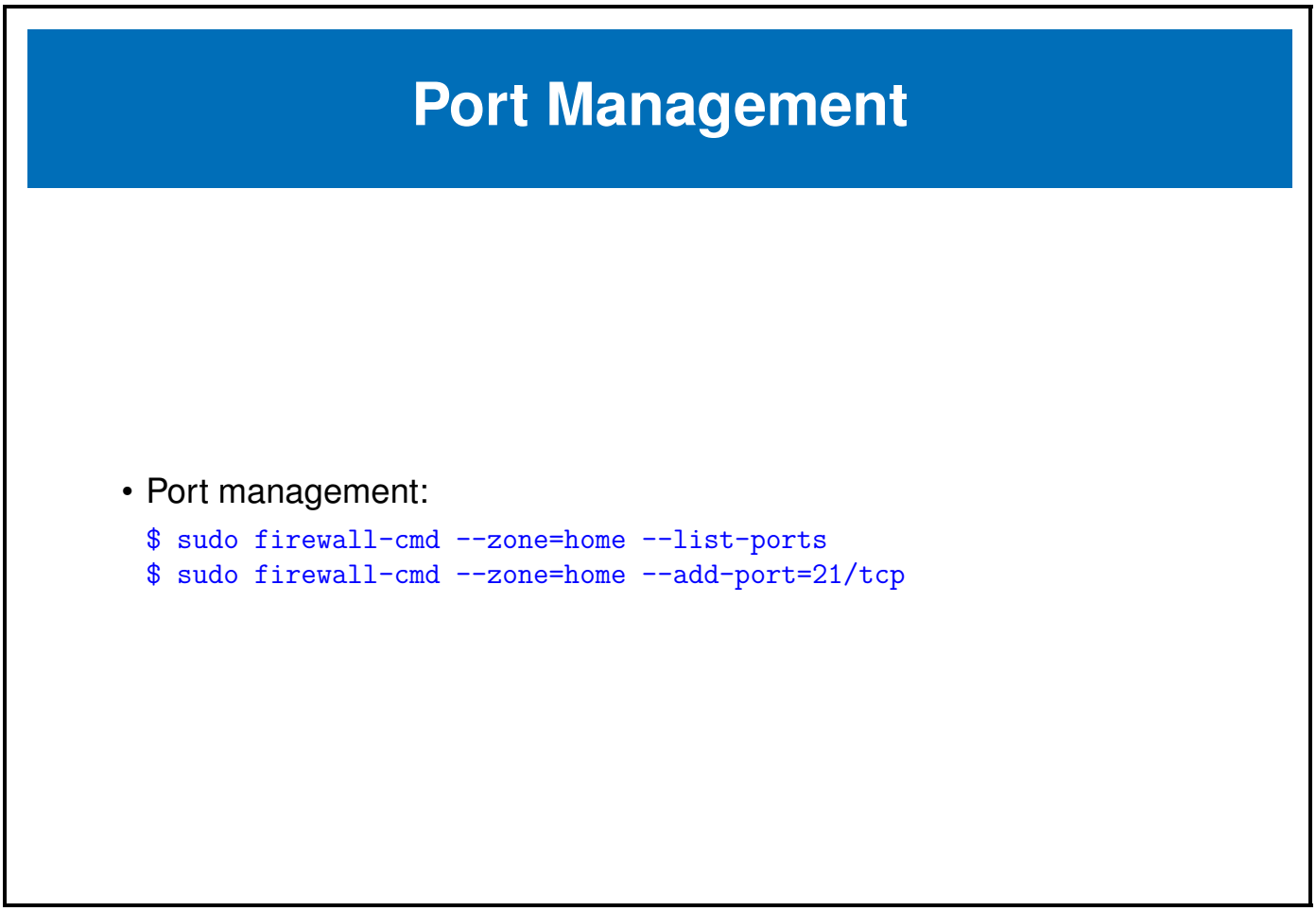
To add a service to a zone:

```
$ sudo firewall-cmd --permanent --zone=home --add-service=dhcp
```

```
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`.



© Copyright the Linux Foundation 2023. All rights reserved.

Port management is very similar to service management:

```
$ sudo firewall-cmd --zone=home --add-port=21/tcp
    success
```

```
$ sudo firewall-cmd --zone=home --list-ports
    21/tcp
```

where by looking at `/etc/services` we can ascertain that port 21 corresponds to **ftp**:

```
$ grep " 21/tcp" /etc/services
    ftp             21/tcp
```


© Copyright the Linux Foundation 2023. All rights reserved.

Network Address Translation (NAT)

Network Address Translation (**NAT**) is a method the firewall uses to change the packet source or destination address as it enters or exits the firewall. There are several types of **NAT**:

- **DNAT** (**Destination Network Address Translation**)
- **SNAT** (**Source Network Address Translation**)
- **MASQUERADE**, a variation of **SNAT**

DNAT, **SNAT** and **MASQUERADE** will be discussed in the next few pages.

© Copyright the Linux Foundation 2023. All rights reserved.

DNAT

DNAT (**Destination Network Address Translation**) alters the end destination (the “to” IP address) and possibly the destination port, as the packet enters the system from the Internet.

- When a packet is sent it will have the public IP address provided by **DNS**
- The public IP address will be that of the firewall
- Changing the destination of the packet is necessary to allow the packet to move through the firewall to the internal network to the desired service

The **DNAT** firewall rule is normally placed in the **pre-routing** chain of the **NAT** table. This would be the **Internet**-facing adapter and the public address used by services like a web server.

Since firewalls are typically pass-through devices (few local services), the packet is examined and the destination port examined to see if there is an applicable rule. Our rule will examine the packet for:

- Destination IP address; it should be that of the firewall
- Destination port number; it will be the port number of the service (http=80)
- The **DNAT** rule will change the destination IP address to the address of the web server behind the firewall.
- The packet will be **forwarded** to the web server behind the firewall
- The web server accepts and processes the packet and sends its reply.

© Copyright the Linux Foundation 2023. All rights reserved.

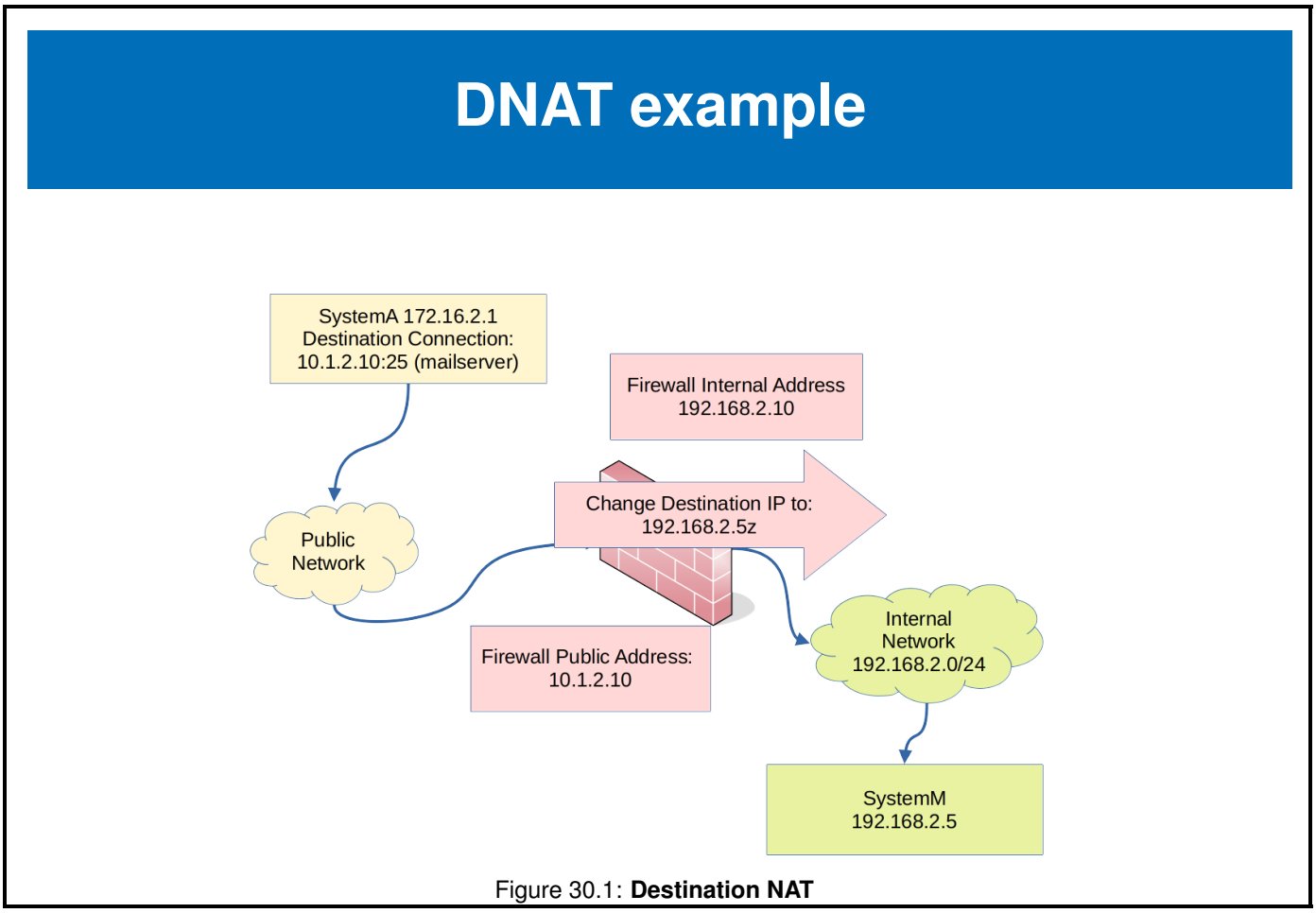


Figure 30.1: **Destination NAT**

The graphic shows a typical path a packet takes from a user system to a service on a system inside a firewall protected network.

- `SystemA` (IP 172.16.2.1) looks up the address for `systemM` (IP 10.1.2.25) for mail delivery.
- The destination port number will be the port number of the service (smtp=25).
- The **DNAT** rule on the firewall will change the destination to the address of the mail server behind the firewall (IP 192.168.2.5).
- The packet will be **forwarded** to the mail server behind the firewall.
- The mail server accepts and processes the packet and sends a reply.

© Copyright the Linux Foundation 2023. All rights reserved.

SNAT and MASQUERADE

Source Network Address Translation (**SNAT**) changes the source (the "from") IP address on the packet as it leaves the system on the way to the Internet.

- Translation is required to protect the addresses used on internal networks from exposure to the Internet
- Most internal networks are private networks and not routable

- If the IP address on the outbound interface is provided by DHCP, the **MASQUERADE** option is used to implement **SNAT**.
- The masquerade option requires more compute resources than **SNAT**.
- If the outbound interface is static, **SNAT** can be used and the public IP address is specified.
- **SNAT** and masquerade are in the **post-routing** chain of the **NAT** table.

© Copyright the Linux Foundation 2023. All rights reserved.

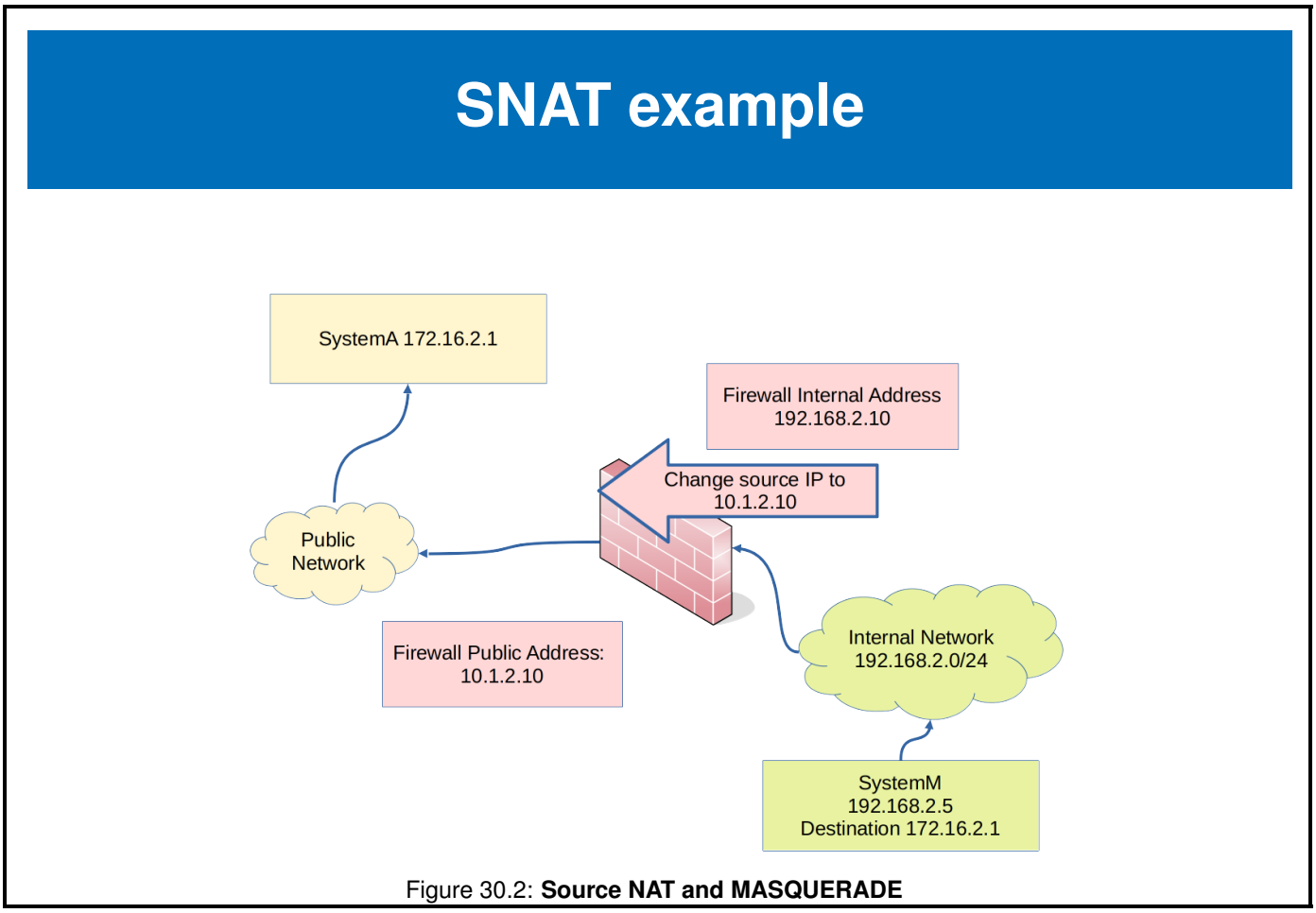


Figure 30.2: **Source NAT and MASQUERADE**

The **SNAT** or **MASQUERADE** firewall rule is normally placed in the **post-routing** chain of the **NAT** table. Normally post-routing is the last examination or modification point as the packet leaves the system on the way to the Internet.

- the destination address of the packet is that of the remote system that initiated the connection.
- the source address of the packet will be changed from the internal address of the service to that of the firewall's public facing adapter.

© Copyright the Linux Foundation 2023. All rights reserved.

30.8 Labs

Exercise 30.1: 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
```

```
/usr/sbin/firewalld
/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 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 30.2: 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
```

```
Usage: firewall-cmd [OPTIONS...]
....
Service Options
  --new-service=<service>
                        Add a new service [P only]
  --delete-service=<service>
                        Delete and existing service [P only]
....
```


© Copyright the Linux Foundation 2023. All rights reserved.

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 30.3: Adding Services to a Zone

Add the **http** and **https** services to the **public zone** and verify that they are currently listed.

Solution 30.3

```
$ sudo firewall-cmd  --zone=public --add-service=http
```

```
success
```

```
$ sudo firewall-cmd  --zone=public --add-service=https
```

```
success
```

```
$ sudo firewall-cmd --list-services --zone=public
```

```
dhcpv6-client http https ssh
```

Note if you had run

```
$ sudo firewall-cmd --reload
$ sudo firewall-cmd --list-services --zone=public
```

```
dhcpv6-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 30.4: 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.

Exercise 30.5: Determine your real IP address

In this exercise we will see the most common usage for **NAT**, consealing the `internal` or `private` addresses of devices behind a **firewall**.

© Copyright the Linux Foundation 2023. All rights reserved.

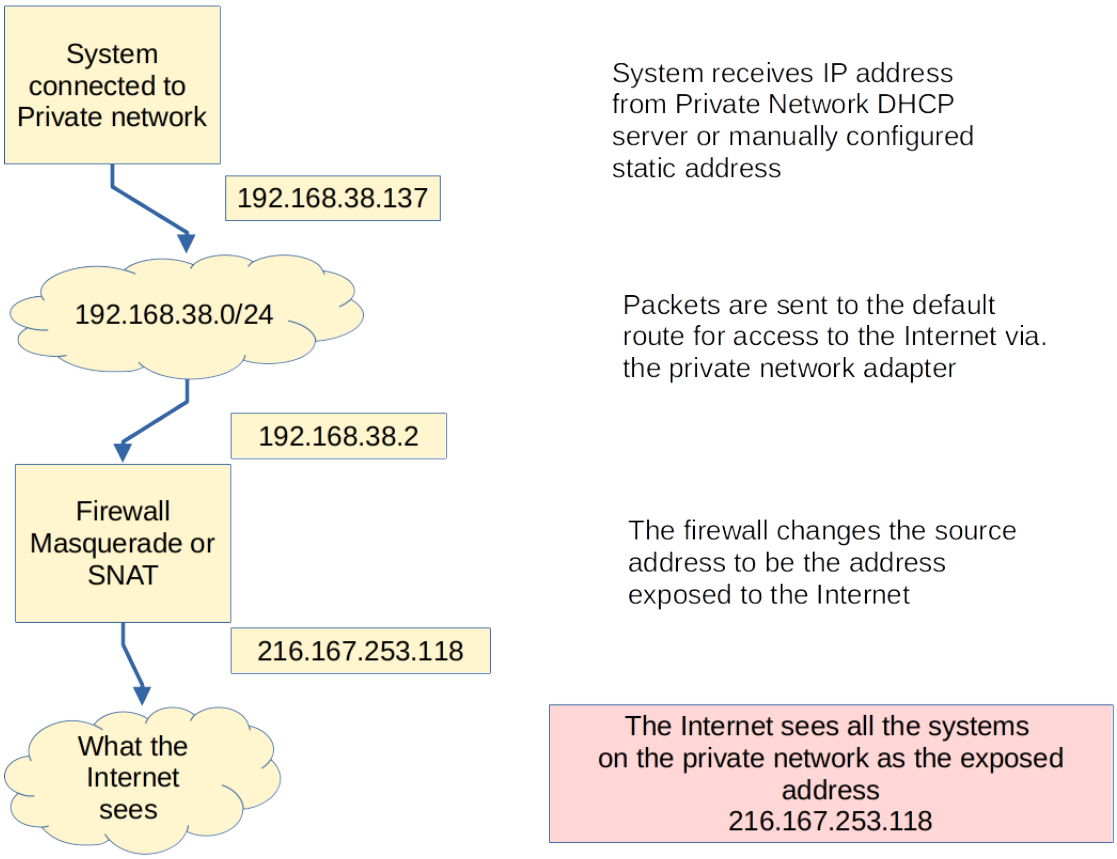


Figure 30.3: **SNAT or MASQUERADE example**

Using the above diagram as an guide determine the IP addresses in use in your system.

1. On your Linux VM, retrieve your network addresses. The **ip** command will work well, and the extra information not needed can be eliminated as shown below.

```
$ sudo ip a | grep " inet "
```

```
    inet 127.0.0.1/8 scope host lo
    inet 192.168.38.137/24 brd 192.168.38.255 scope global dynamic noprefixroute ens33
```

   The addresses presented are the **loopback** and the IP address of the Virtual Machine.

2. The address of the VM is in the subnet of **192.168.24.0/24** and is a **non-routeable** address on the **internet**. See the link below for additional information on **Address Allocation for Private Internets**
   https://datatracker.ietf.org/doc/html/rfc1918
   To find the next IP address that will connect out client to the next hop, examine the **default route**.

```
    default via 192.168.38.2 dev ens33 proto dhcp metric 100
```

   We can see our next hop is the system with the IP address of **192.168.38.2**.

3. The system **192.168.38.2** is unavailable for us to login to as it is the real firewall for this environment. We can extract how the world sees our system.
   Via command line:

```
$ sudo lynx -dump http://whatismyip.akamai.com  | awk '{print $1}'
```


© Copyright the Linux Foundation 2023. All rights reserved.

```
216.167.253.118
```

Or select one of many graphical options via a Goolge search for:
what is my ip address

Notice the address used to communicate with the outside world is not one of the private addresses and all the machines on the private network look like one IP Address to the outside world.
The action of changing the private IP Addresses to the single address is an example of **Source Network Translation** or **Masquerade**

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 31

System Init: systemd history and customization

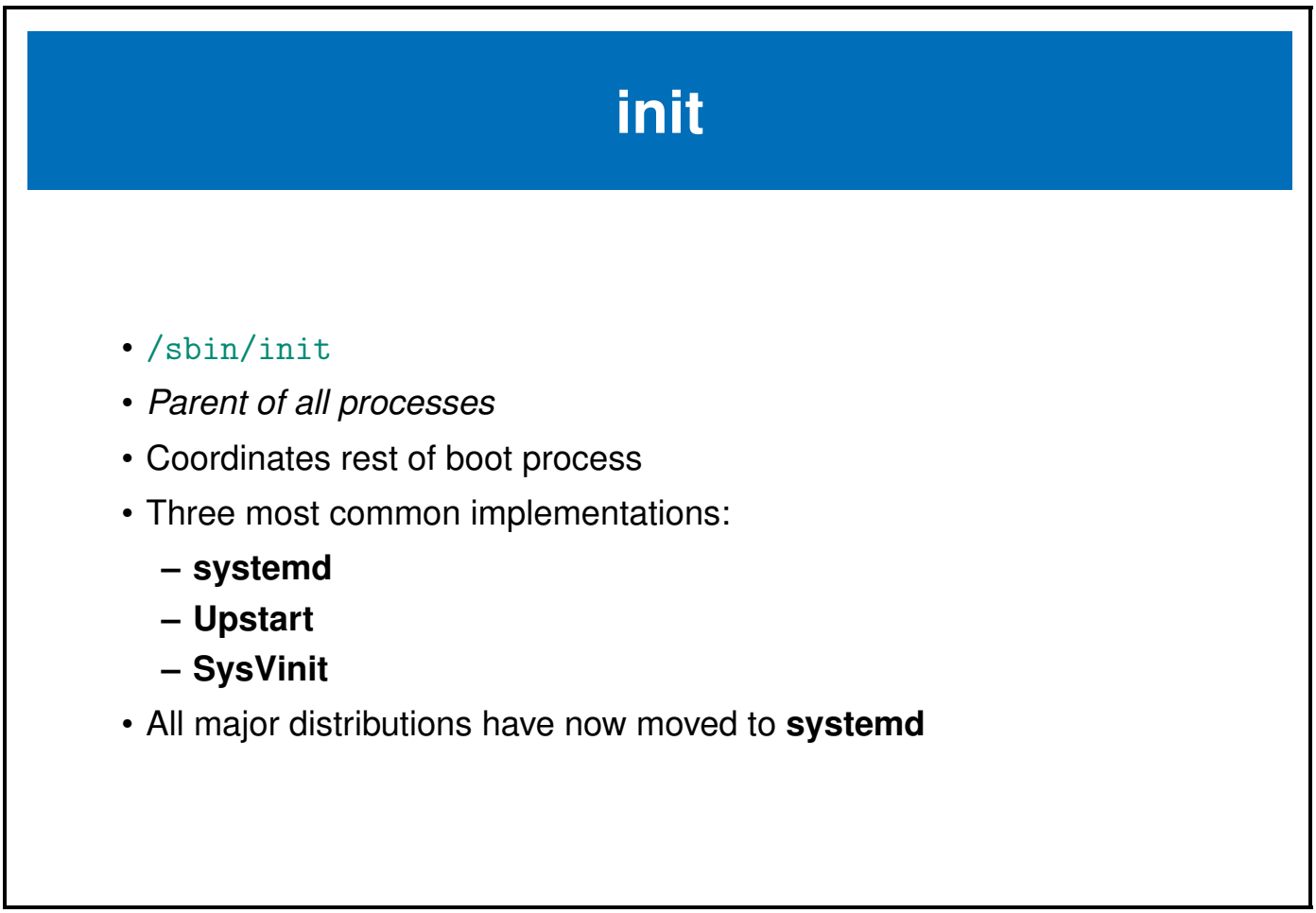
31.1 The init Process 486
31.2 Startup Alternatives 487
31.3 systemd 488
31.4 systemctl 490
31.5 Labs 491

Learning Objectives

By the end of this session, you should be able to:

- Understand how **systemd** arose to replace **SysVinit** and **Upstart**.
- Explain the main features of **systemd**.
- Use **systemctl** to configure and control services with **systemd**.
- See how user **systemd units** and drop-in files function.

31.1 The init Process



`/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.

© Copyright the Linux Foundation 2023. All rights reserved.

31.2 Startup Alternatives

Startup Alternatives

- **Upstart**
  - Developed by **Ubuntu** and first included in 2006
  - Adopted in **Fedora** 9 (in 2008) and in **RHEL 6** and its clones
  - Was also used in various embedded and mobile devices
- **systemd**
  - **Fedora** 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 chapter, we will concentrate just on **systemd**. For the most part, we will ignore **SysVinit** as well as **Upstart** (which is no longer used in any significant way).

© Copyright the Linux Foundation 2023. All rights reserved.

31.3 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
- Provides detailed start and stop functionality, including similar abilities to **cron**.

The **systemd** system and session manager for **Linux** is now dominant in most major distributions.

Instead of **bash** scripts, **systemd** uses `.service` files.

Granular logging is provided by default using the name of the **unit**.

Tracking and creating user processes without root privileges is another powerful ability of **systemd**.

© Copyright the Linux Foundation 2023. All rights reserved.

systemd Configuration File Locations

systemd allows for multiple locations for its **unit** files, allowing for processes running under a non-root user, and inheritance from system **unit** files. Here are some typical locations (but may vary by distribution):

system unit files:

- `/etc/systemd/system`
- `/lib/systemd/system`

user unit files:

- `/etc/systemd/user`
- `~/.config/systemd/user`

Please Note

On most systems, `/lib/systemd/` is a symlink to `/usr/lib/systemd/`.

Here is an example of a standard **system** unit file from **Debian**:

/lib/systemd/system/dbus.service

```
[Unit]
Description=D-Bus System Message Bus
Documentation=man:dbus-daemon(1)
Requires=dbus.socket

[Service]
ExecStart=/usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation
↪  --syslog-only
ExecReload=/usr/bin/dbus-send --print-reply --system --type=method_call --dest=org.freedesktop.DBus /
↪  org.freedesktop.DBus.ReloadConfig
OOMScoreAdjust=-900
```

Unit files can be quite complex. In this example, you can see a way to include a link to documentation and different methods for starting and restarting the service.

© Copyright the Linux Foundation 2023. All rights reserved.

31.4 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.

In a similar fashion, you can see **user** units:

```
$ systemctl list-units --user 'xdg*'
```

```
    UNIT                          LOAD   ACTIVE SUB     DESCRIPTION
    xdg-desktop-portal-gtk.service loaded active running Portal service (GTK+/GNOME implementation)
    xdg-desktop-portal.service     loaded active running Portal service
    xdg-document-portal.service    loaded active running flatpak document portal service
    xdg-permission-store.service   loaded active running sandboxed app permission store

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.
4 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.
```

Please Note

We limited our output to only units starting with *xdg* under a desktop `Xorg` session in the command above because there are a large number of user units too.

© Copyright the Linux Foundation 2023. All rights reserved.

31.5 Labs

Video Demonstration Resources

using_systemctl_demo.mp4

Exercise 31.1: Modify an Existing Service with systemd

Often we want to modify the behavior of an existing service rather than creating a new one. **systemd** can handle this by completely replacing or disabling a built-in service, but also by only modifying certain parameters of an existing service, called a **drop-in** via an `.override` file or `.config` directory. An added advantage of using this method – package upgrades won't revert your changes.

We are going to modify the `Description` for the **unit** responsible for login (`getty`) on the first console (`tty1`) and then change it back.

We are assuming you are logged-in to your workstation either via the graphical interface or ssh. But a safety precaution, verify that you are not logged in on this terminal:

```
$ tty
```

Very Important

If you see the result of this command as `/dev/tty1`, it's probably best to skip this lab.

1. First, check the status of the `getty@tty1.service`:

   ```
   $ systemctl status getty@tty1.service
   ```

   ```
   $ systemctl status getty@tty1.service
   * getty@tty1.service - Getty on tty1
        Loaded: loaded (/lib/systemd/system/getty@.service; enabled; vendor preset: enabled)
        Active: active (running) since Thu 2023-03-09 12:58:42 EST; 2 months 7 days ago
          Docs: man:agetty(8)
                man:systemd-getty-generator(8)
                http://0pointer.de/blog/projects/serial-console.html
      Main PID: 621 (agetty)
         Tasks: 1 (limit: 4656)
        Memory: 200.0K
           CPU: 2ms
        CGroup: /system.slice/system-getty.slice/getty@tty1.service
                `-621 /sbin/agetty -o -p -- \u --noclear tty1 linux

   Warning: some journal files were not opened due to insufficient permissions.
   ```

2. Next, let's change the description of the service. Copy the **Description=** line and make it say something different. In the example, we are changing the dynamic output to **first terminal**.

   ```
   $ sudo systemctl edit getty@tty1.service
   ```

   ```
   ### Editing /etc/systemd/system/getty@tty1.service.d/override.conf
   ### Anything between here and the comment below will become the new contents of the file

   [Unit]
   Description=Getty on the first terminal

   ### Lines below this comment will be discarded
   ```


© Copyright the Linux Foundation 2023. All rights reserved.

```
### /lib/systemd/system/getty@.service
# #  SPDX-License-Identifier: LGPL-2.1-or-later
# #
# #  This file is part of systemd.
# #
# #  systemd is free software; you can redistribute it and/or modify it
# #  under the terms of the GNU Lesser General Public License as published by
# #  the Free Software Foundation; either version 2.1 of the License, or
# #  (at your option) any later version.
```

3. Now, verify the contents of /etc/systemd/system/getty@tty1.service.d/override.conf
This should represent the change that you just made - just two lines with no newline at the end.

```
$ cat /etc/systemd/system/getty@tty1.service.d/override.conf
```

```
[Unit]
Description=Getty on the first terminal<YOUR PROMPT HERE>
```

4. Reload the daemon to pick up our changes, and the check the results:

```
$ sudo systemctl daemon-reload
$ systemctl status getty@tty1.service
```

```
* getty@tty1.service - Getty on the first terminal
     Loaded: loaded (/lib/systemd/system/getty@.service; enabled; vendor preset: enabled)
    Drop-In: /etc/systemd/system/getty@tty1.service.d
             `-override.conf
     Active: active (running) since Thu 2023-03-09 12:58:42 EST; 2 months 7 days ago
       Docs: man:agetty(8)
             man:systemd-getty-generator(8)
             http://0pointer.de/blog/projects/serial-console.html
   Main PID: 621 (agetty)
      Tasks: 1 (limit: 4656)
     Memory: 200.0K
        CPU: 2ms
     CGroup: /system.slice/system-getty.slice/getty@tty1.service
             `-621 /sbin/agetty -o -p -- \u --noclear tty1 linux

Warning: some journal files were not opened due to insufficient permissions.
```

5. Finally, remove the file we created to bring things back to normal.

```
$ sudo rm  /etc/systemd/system/getty@tty1.service.d/override.conf
$ systemctl status getty@tty1.service
```

```
* getty@tty1.service - Getty on tty1
     Loaded: loaded (/lib/systemd/system/getty@.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2023-03-09 12:58:42 EST; 2 months 7 days ago
       Docs: man:agetty(8)
             man:systemd-getty-generator(8)
             http://0pointer.de/blog/projects/serial-console.html
   Main PID: 621 (agetty)
      Tasks: 1 (limit: 4656)
     Memory: 200.0K
        CPU: 2ms
     CGroup: /system.slice/system-getty.slice/getty@tty1.service
             `-621 /sbin/agetty -o -p -- \u --noclear tty1 linux
```


© Copyright the Linux Foundation 2023. All rights reserved.

```
Warning: some journal files were not opened due to insufficient permissions.
```


© Copyright the Linux Foundation 2023. All rights reserved.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 32

Backup and Recovery Methods

32.1 Backup Basics 496
32.2 Backup vs Archive 498
32.3 Backup Methods and Strategies 500
32.4 tar 503
32.5 Compression: gzip, bzip2 and xz and Backups 506
32.6 dd 507
32.7 rsync 508
32.8 Backup Programs ** 509
32.9 Labs 510

Learning Objectives

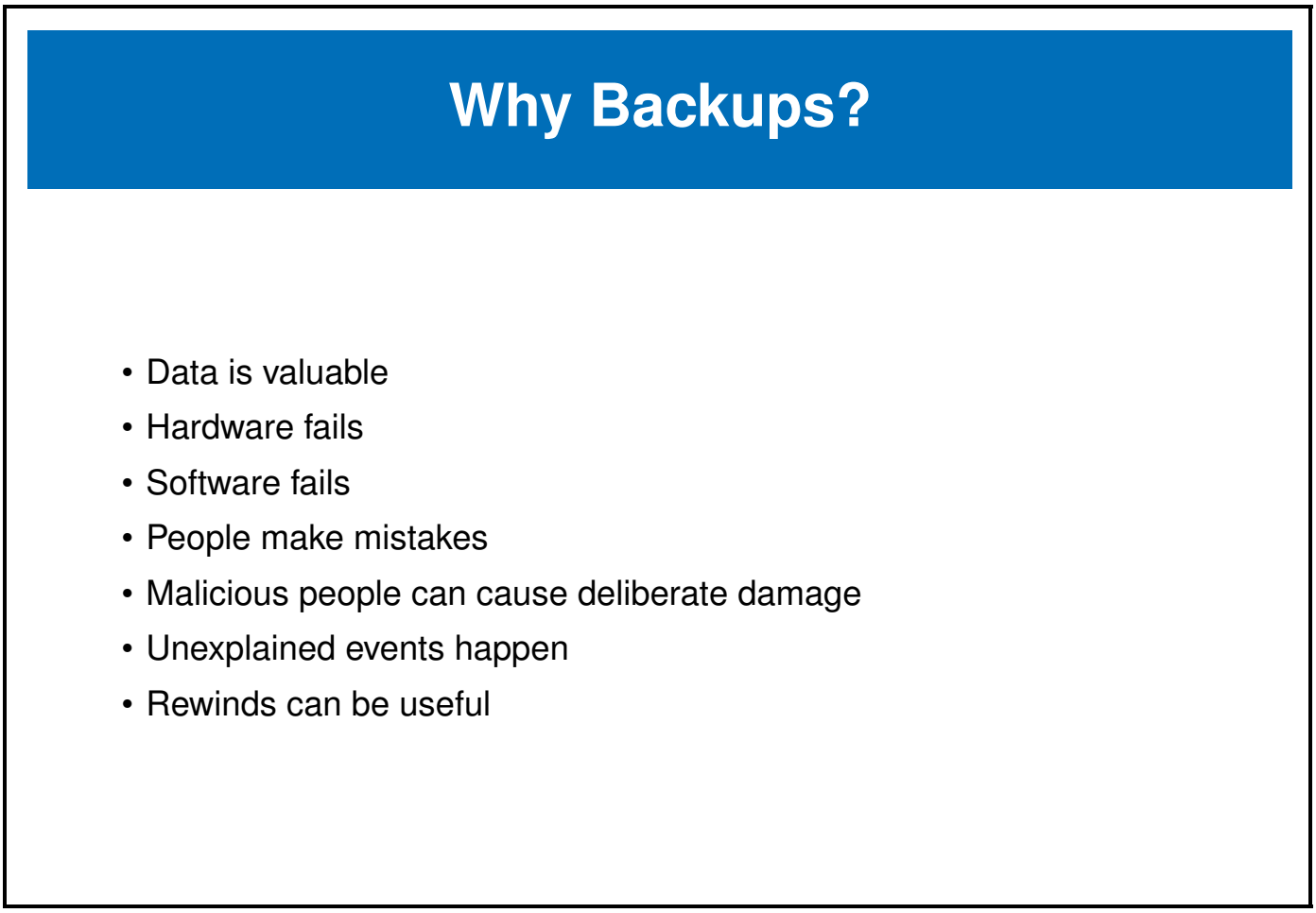
By the end of this session, you should be able to:

- Identify and prioritize data that needs backup.
- Understand the difference between backing up and archiving data.
- Efficiently employ different kinds of backup and restore methods, depending on the situation.
- Use a variety of compression methods such as **xz** and **gzip**, and combine them efficiently with the **tar** archiving program.
- Use **dd** for raw data copying, and **rsync** for efficient backup and synchronization across multiple machines.
- Describe some of the most well-known backup programs.

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.

32.1 Backup Basics



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.

© Copyright the Linux Foundation 2023. All rights reserved.

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).

© Copyright the Linux Foundation 2023. All rights reserved.

32.2 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 backup 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.

© Copyright the Linux Foundation 2023. All rights reserved.

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** (**L**inear **T**ape **O**pen) 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** (**N**etwork **A**ttached **S**torage) 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.

© Copyright the Linux Foundation 2023. All rights reserved.

32.3 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.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

32.4 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`.

© Copyright the Linux Foundation 2023. All rights reserved.
THE LINUX FOUNDATION | Training & Certification

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.

© Copyright the Linux Foundation 2023. All rights reserved.
THE LINUX FOUNDATION Training & Certification

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**.

© Copyright the Linux Foundation 2023. All rights reserved.
THE LINUX FOUNDATION | Training & Certification

32.5 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:

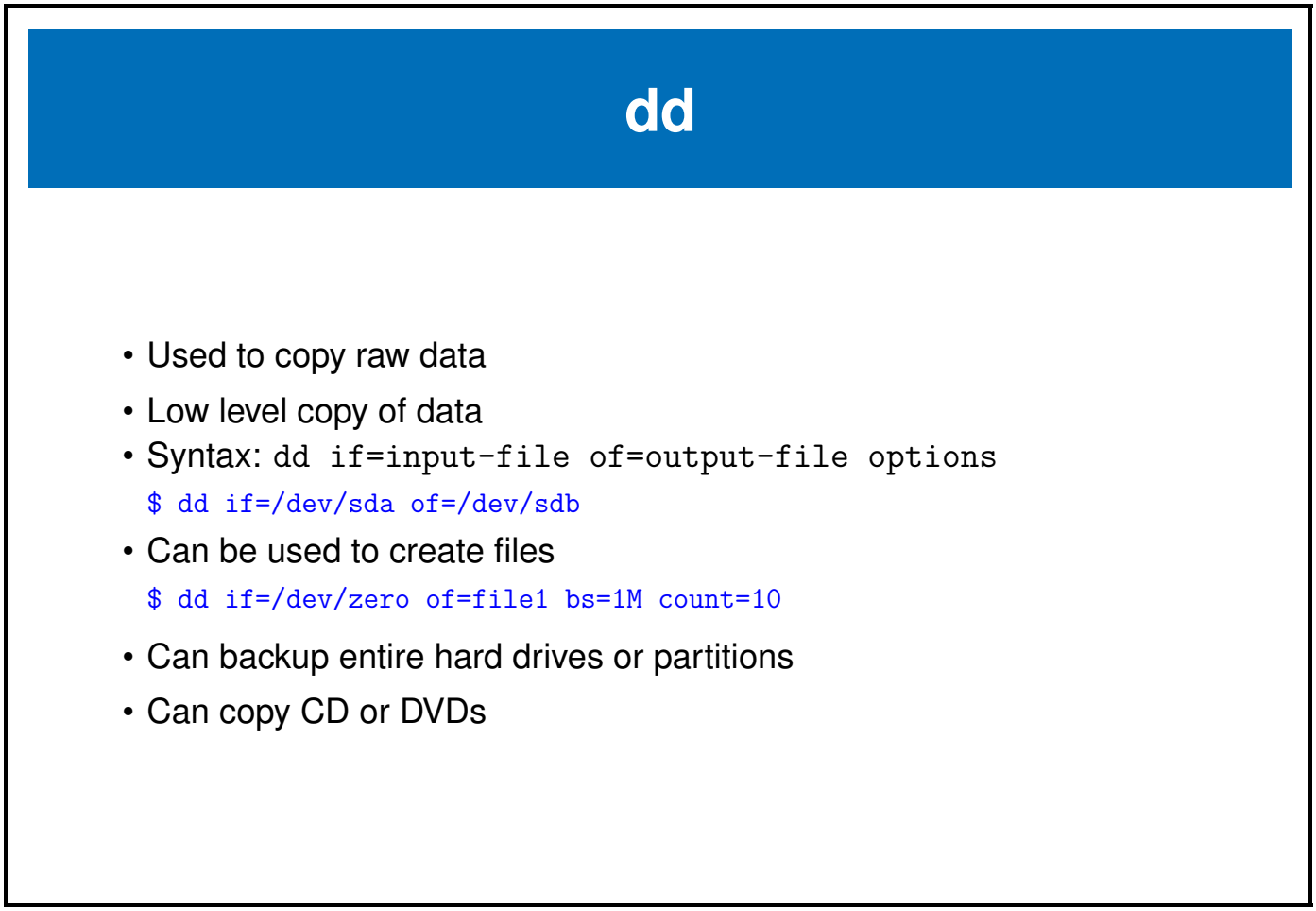
```
$ tar xvf source.tar.gz
```

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.

© Copyright the Linux Foundation 2023. All rights reserved.

32.6 dd



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
```


© Copyright the Linux Foundation 2023. All rights reserved.

32.7 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 (**r**emote **sync**hronize) 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.

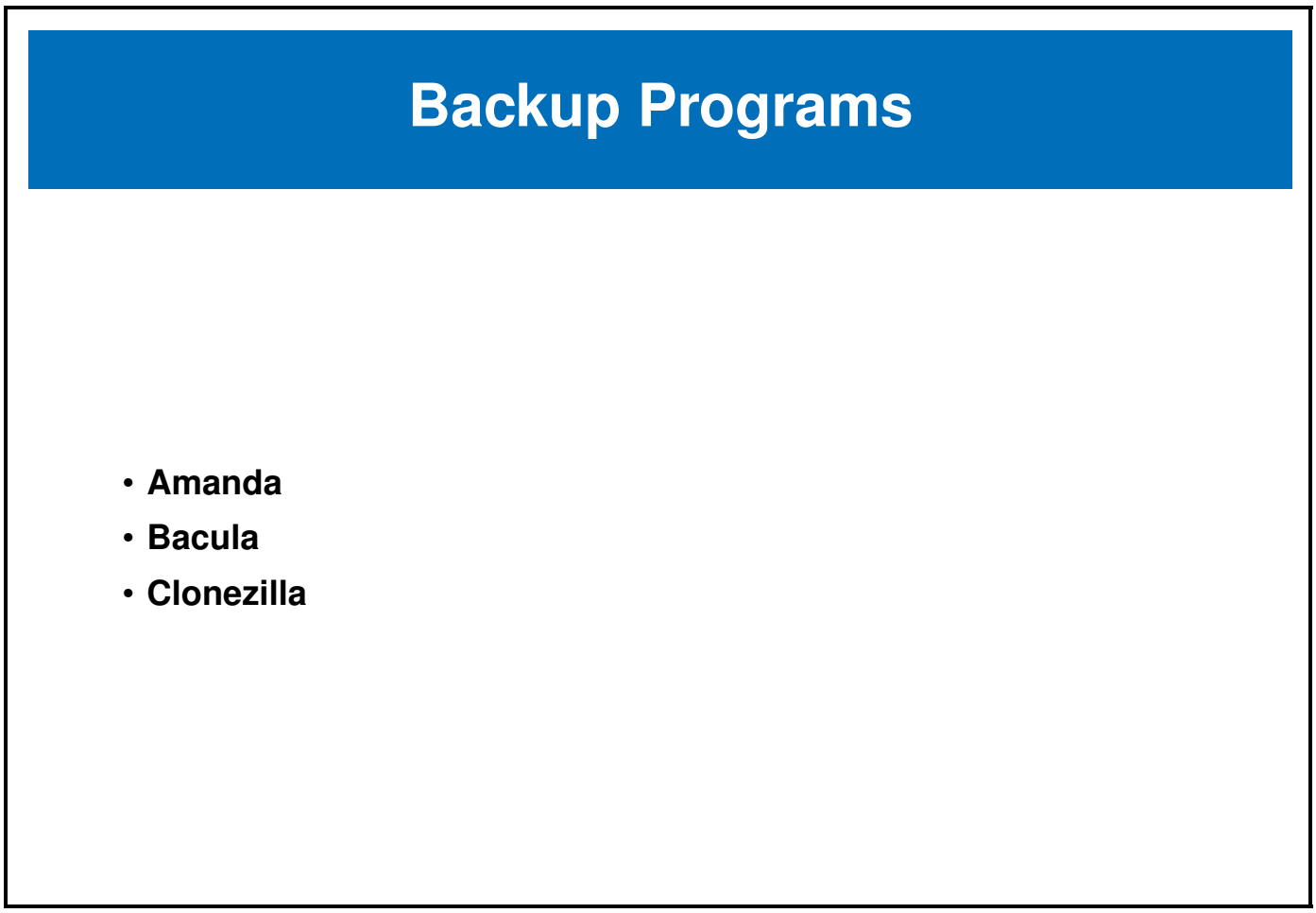
rsync is very clever; it checks local files against remote files in small chunks, and it is very efficient in that when copying one directory to a similar directory, only the differences are copied over the network with the first directory. One often uses the `-r` option which causes **rsync** to recursively 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.

© Copyright the Linux Foundation 2023. All rights reserved.

32.8 Backup Programs **



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.

- **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**), filesystem types, and boot loaders.

© Copyright the Linux Foundation 2023. All rights reserved.

32.9 Labs

Exercise 32.1: 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 32.1

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
```
```
55M        /usr/include
```

2.
```
$ ls -lh include.tar.*
```
```
c8:/tmp/backup>ls -lh
total 17M
-rw-rw-r-- 1 coop coop 5.3M Jul 18 08:17 include.tar.bz2
-rw-rw-r-- 1 coop coop 6.7M Jul 18 08:16 include.tar.gz
-rw-rw-r-- 1 coop coop 4.5M Jul 18 08:18 include.tar.xz
```

3.
```
$ tar tvf include.tar.xz
```
```
qdrwxr-xr-x root/root         0 2014-10-29 07:04 include/
-rw-r--r-- root/root     42780 2014-08-26 12:24 include/unistd.h
-rw-r--r-- root/root       957 2014-08-26 12:24 include/re_comp.h
-rw-r--r-- root/root     22096 2014-08-26 12:24 include/regex.h
-rw-r--r-- root/root      7154 2014-08-26 12:25 include/link.h
.....
```
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
```
```
include/
include/unistd.h
include/re_comp.h
include/regex.h
include/link
.....
$ diff -qr include /usr/include
```


© Copyright the Linux Foundation 2023. All rights reserved.

Exercise 32.2: Using rsync for Backup

1. Using **rsync**, we will again create a complete copy of `/usr/include` in your backup directory:

```
$ rm -rf include
$ rsync -av /usr/include .
```

```
sending incremental file list
include/
include/FlexLexer.h
include/_G_config.h
include/a.out.h
include/aio.h
.....
```

2. Let's run the command a second time and see if it does anything:

```
$ rsync -av /usr/include .
```

```
sending incremental file list

sent 127398 bytes  received 188 bytes  255172.00 bytes/sec
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
```

```
sending incremental file list
...
```

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 .
```

```
sending incremental file list
include/
deleting include/include/xen/privcmd.h
deleting include/include/xen/evtchn.h
....
deleting include/include/FlexLexer.h
deleting include/include/

sent 127401 bytes  received 191 bytes  85061.33 bytes/sec
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 .
```

```
sending incremental file list
include/
include/xen/
include/xen/evtchn.h
```


© Copyright the Linux Foundation 2023. All rights reserved.

```
include/xen/privcmd.h

sent 127412 bytes  received 202 bytes  255228.00 bytes/sec
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**:

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.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 33

Linux Security Modules

33.1 Linux Security Modules 514
33.2 SELinux 516
33.3 AppArmor 531
33.4 Labs 537

Learning Objectives

By the end of this session, you should be able to:

- Understand how the Linux Security Module framework works and how it is deployed.
- List the various LSM implementations available.
- Delineate the main features of SELinux.
- Explain the different modes and policies available.
- Grasp the importance of contexts and how to get and set them.
- Know how to use the important SELinux utility programs.
- Gain some familiarity with AppArmor.

33.1 Linux Security Modules

What are Linux Security Modules?

- An **LSM** (**L**inux **S**ecurity **M**odule) 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** (**L**inux **S**ecurity **M**odule).

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.

© Copyright the Linux Foundation 2023. All rights reserved.

Main LSM Choices

- **SELinux**: https://selinuxproject.org/page/Main_Page
- **AppArmor**: https://apparmor.net
- **Smack**: http://schaufler-ca.com
- **TOMOYO**: https://tomoyo.osdn.jp
- **Yama**: https://kernel.org/doc/html/latest/admin-guide/LSM/Yama.html

Use of Simultaneous LSMs

Since 2019 it has been possible to combine (stack) **LSM**s 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. **LSM**s 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.

© Copyright the Linux Foundation 2023. All rights reserved.

33.2 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

SELinux was originally developed by the United States **NSA** (**N**ational **S**ecurity **A**dministration) 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**.

A **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.

© Copyright the Linux Foundation 2023. All rights reserved.

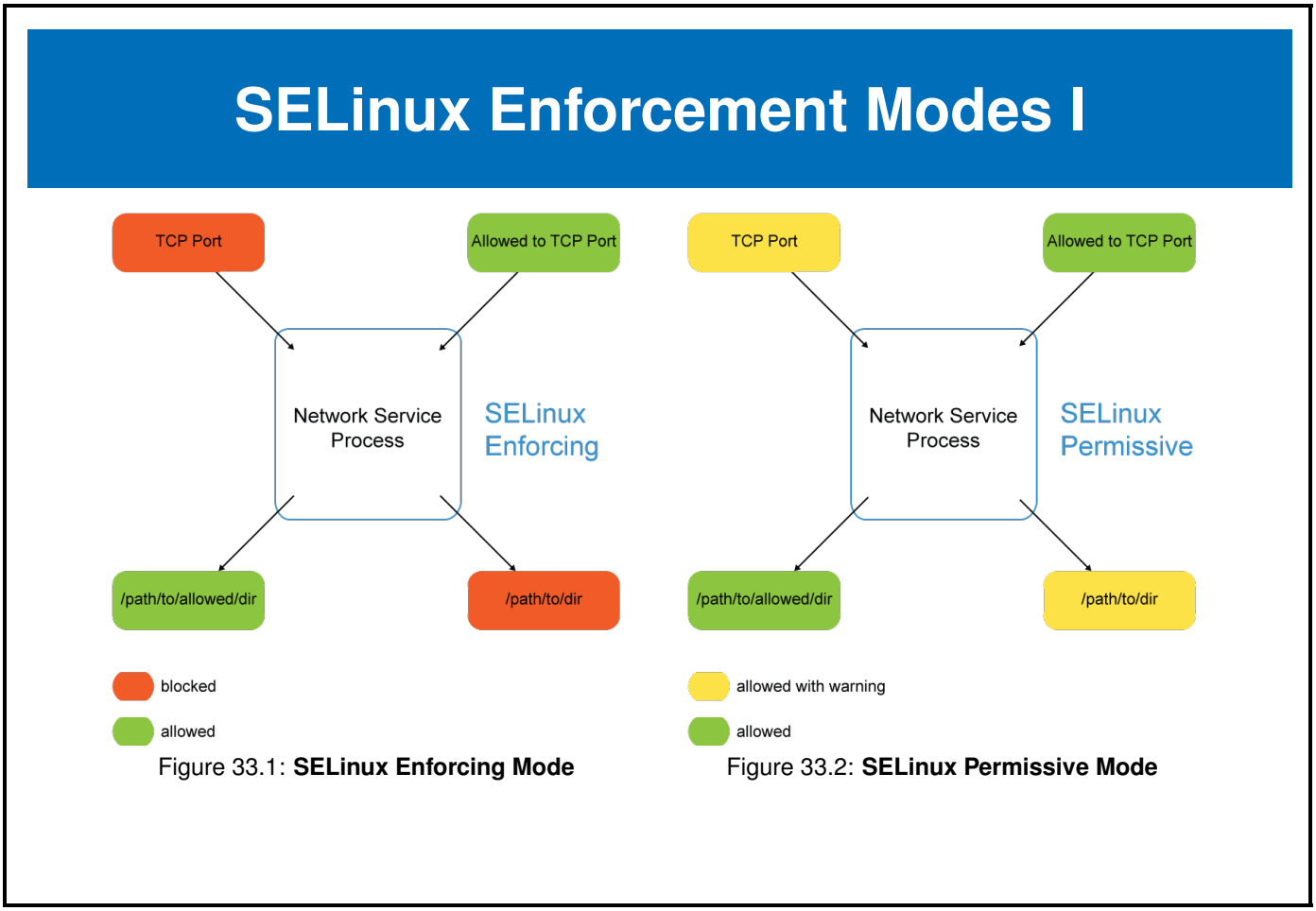


Figure 33.1: **SELinux Enforcing Mode**

Figure 33.2: **SELinux Permissive Mode**

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.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

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 **M**ulti-**L**evel **S**ecurity 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]`.

© Copyright the Linux Foundation 2023. All rights reserved.

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
```


© Copyright the Linux Foundation 2023. All rights reserved.

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
```

```
-rw-rw-r--. dog dog unconfined_u:object_r:user_home_t:s0 somefile
```

```
$ chcon -t etc_t somefile
$ ls -Z
```

```
-rw-rw-r--. dog dog unconfined_u:object_r:etc_t:s0   somefile
```

```
$ ls -Z
```

```
-rw-rw-r--. dog dog unconfined_u:object_r:etc_t:s0   somefile
-rw-rw-r--. dog dog unconfined_u:object_r:user_home_t:s0 somefile1
```

```
$ chcon --reference somefile somefile1
$ ls -Z
```

```
-rw-rw-r--. dog dog unconfined_u:object_r:etc_t:s0   somefile
-rw-rw-r--. dog dog unconfined_u:object_r:etc_t:s0   somefile1
```


© Copyright the Linux Foundation 2023. All rights reserved.
THE LINUX FOUNDATION Training & Certification

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

© Copyright the Linux Foundation 2023. All rights reserved.

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
```

```
LABEL PID TTY STAT TIME COMMAND
system_u:system_r:init_t:s0 1 ? Ss 0:04 /usr/lib/systemd/systemd --switched-root ...
system_u:system_r:kernel_t:s0 2 ? S 0:00 [kthreadd]
...
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 2305 ? D 0:00 sshd: jimih@pts/0
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 2306 pts/0 Ss 0:00 -bash
...
system_u:system_r:httpd_t:s0 7490 ? Ss 0:00 /usr/sbin/httpd -DFOREGROUND
system_u:system_r:httpd_t:s0 7491 ? S 0:00 /usr/sbin/httpd -DFOREGROUND
...
```

```
$ ls -Z /home/ /tmp/
```

```
/home/:
drwx------. jimih jimih unconfined_u:object_r:user_home_dir_t:s0 jimih
/tmp/:
-rwx------. root root system_u:object_r:initrc_tmp_t:s0 ks-script-c4ENhg
drwx------. root root system_u:object_r:tmp_t:s0 systemd-private-0ofSv0
-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.

Context Inheritance

- New files inherit context from parent directory
- When moving files, source directory context preserved
- Can cause problems if context is not modified

© Copyright the Linux Foundation 2023. All rights reserved.

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
```

```
-rw-rw-r--. jimih jimih unconfined_u:object_r:user_tmp_t:s0 tmpfile
```

```
$ cd
$ touch homefile
$ ls -Z homefile
```

```
-rw-rw-r--. jimih jimih unconfined_u:object_r:user_home_t:s0 homefile
```

```
$ mv /tmp/tmpfile .
$ ls -Z
```

```
-rw-rw-r--. jimih jimih unconfined_u:object_r:user_home_t:s0 homefile
-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.

© Copyright the Linux Foundation 2023. All rights reserved.

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
```

```
-rw-rw-r--. jimih jimih unconfined_u:object_r:user_home_t:s0 homefile
-rw-rw-r--. jimih jimih unconfined_u:object_r:user_tmp_t:s0 tmpfile
```

```
$ restorecon -Rv /home/jimih
```

```
restorecon reset /home/jimih/tmpfile context \
          unconfined_u:object_r:user_tmp_t:s0->unconfined_u:object_r:user_home_t:s0
```

```
$ ls -Z
```

```
-rw-rw-r--. jimih jimih unconfined_u:object_r:user_home_t:s0 homefile
-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`.

© Copyright the Linux Foundation 2023. All rights reserved.

semanage

- Configure default context for a new directory
- Changes only default settings, not existing ones
- **restorecon** must be called afterwards to change existing contexts

© Copyright the Linux Foundation 2023. All rights reserved.

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
```

```
...
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
```

```
...
drwxr-xr-x. root root unconfined_u:object_r:default_t:s0 virtualHosts
```

```
[root@rhel7 /]# restorecon -RFv /virtualHosts
```

```
restorecon reset /virtualHosts context \
          unconfined_u:object_r:default_t:s0->system_u:object_r:httpd_sys_content_t:s0
```

```
[root@rhel7 /]# ls -Z
```

```
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**.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

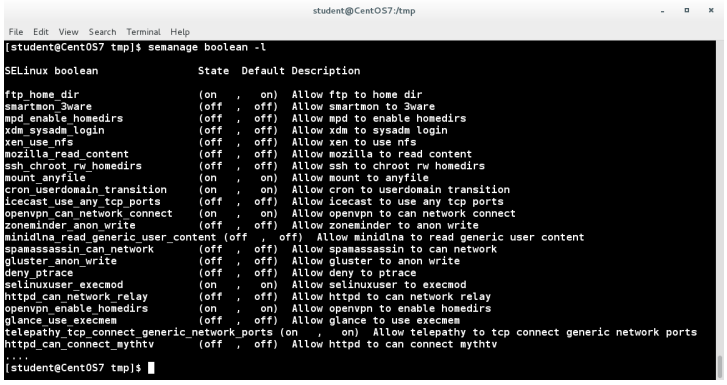


Figure 33.3: **semanage**

© Copyright the Linux Foundation 2023. All rights reserved.

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:

```
[root@rhel7 ~]# echo 'File created at /root' > rootfile
[root@rhel7 ~]# mv rootfile /var/www/html/
[root@rhel7 ~]# wget -O - localhost/rootfile
--2014-11-21 13:42:04--  http://localhost/rootfile
Resolving localhost (localhost)... ::1, 127.0.0.1
Connecting to localhost (localhost)|::1|:80... connected.
HTTP request sent, awaiting response... 403 Forbidden
2014-11-21 13:42:04 ERROR 403: Forbidden.

[root@rhel7 ~]# tail /var/log/messages
Nov 21 13:42:04 rhel7 setroubleshoot: Plugin Exception restorecon
Nov 21 13:42:04 rhel7 setroubleshoot: SELinux is preventing /usr/sbin/httpd from getattr access on the file .
For complete SELinux messages. run sealert -l d51d34f9-91d5-4219-ad1e-5531e61a2dc3
Nov 21 13:42:04 rhel7 python: SELinux is preventing /usr/sbin/httpd from getattr access on the file .
.....
Do allow this access for now by executing:
# grep httpd /var/log/audit/audit.log | audit2allow -M mypol
# semodule -i mypol.pp

Additional Information:
Source Context system_u:system_r:httpd_t:s0
Target Context unconfined_u:object_r:admin_home_t:s0
Target Objects [ file ]
Source httpd
Source Path /usr/sbin/httpd
......
```


© Copyright the Linux Foundation 2023. All rights reserved.

33.3 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** **D**iscretionary **A**ccess **C**ontrol (**DAC**) model by providing **M**andatory **A**ccess **C**ontrol (**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.

© Copyright the Linux Foundation 2023. All rights reserved.

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**

© Copyright the Linux Foundation 2023. All rights reserved.

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
```

```
   apparmor module is loaded.
  25 profiles are loaded.
  25 profiles are in enforce mode.
     /sbin/dhclient
  ....
```

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 libvirtd
```

```
  root        787  0.0  0.9 527200 35936 ?        Ssl  10:54   0:00 /usr/sbin/libvirtd
  student    3346  0.0  0.0  13696  2204 pts/16   S+   11:42   0:00 grep --color=auto libvirtd
```


© Copyright the Linux Foundation 2023. All rights reserved.

Modes and Profiles

- Modes:
  - **Enforce Mode** Restricted actions are prevented and reported. Set with **aa-enforce**
  - **Complain** Polices not enforced, but policy violations are reported. Set with **aa-complain**
- **Profiles** configured in `/etc/apparmor.d` give details for utilities and software packages

© Copyright the Linux Foundation 2023. All rights reserved.

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
```

```
abstractions              usr.lib.dovecot.anvil        usr.lib.telepathy
apache2.d                 usr.lib.dovecot.auth         usr.sbin.avahi-daemon
bin.ping                  usr.lib.dovecot.config       usr.sbin.cups-brows
....
```

Full documentation on what can go in these files can be obtained by doing `man apparmor.d`.

AppArmor Utilities

Table 33.1: **AppArmor Utilities**

Program	Use
apparmor_status	Show status of all profiles and processes with profiles
apparmor_notify	Show a summary for **AppArmor** log messages
complain	Set a specified profile to complain mode.
enforce	Set a specified profile to enforce mode.
disable	Unload a specified profile from the current kernel and prevent from being loaded on system startup.
logprof	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.
easyprof	Help set up a basic **AppArmor** profile for a program.

© Copyright the Linux Foundation 2023. All rights reserved.

AppArmor has quite a few administrative utilities for monitoring and control. For example, on an **openSUSE** system:

```
$ rpm -qil apparmor-utils | grep bin
    /usr/bin/aa-easyprof
    /usr/sbin/aa-audit
    /usr/sbin/aa-autodep
    /usr/sbin/aa-cleanprof
    /usr/sbin/aa-complain
    /usr/sbin/aa-decode
    /usr/sbin/aa-disable
    /usr/sbin/aa-enforce
    /usr/sbin/aa-exec

    /usr/sbin/complain
    /usr/sbin/decode
    /usr/sbin/disable
    /usr/sbin/enforce
    ....
```

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
    lrwxrwxrwx 1 root root    11 Nov 11 13:02 /usr/sbin/complain -> aa-complain*
    linux-llgn:/etc/apparmor.d #
```


© Copyright the Linux Foundation 2023. All rights reserved.

33.4 Labs

Exercise 33.1: 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
   ```

   ```
   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
   ```

   ```
                                   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
   ```

   ```
   -rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 file1.html
   -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
   ```

   ```
   file2
   ```

Exercise 33.2: Exploring apparmor security

© Copyright the Linux Foundation 2023. All rights reserved.

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** 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 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.
```

   We can see **ping** has a profile that is loaded and enabled for enforcement.

© Copyright the Linux Foundation 2023. All rights reserved.

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:

```
Adding capability net_raw, to profile.

Profile:        /bin/ping-x
Network Family: inet
Socket Type:    raw
```


© Copyright the Linux Foundation 2023. All rights reserved.

```
 [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.
```

6. View the created profile, which has been stored in `/etc/appamor.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,

  /bin/ping-x mr,
  /lib/x86_64-linux-gnu/ld-*.so mr,

}
```

7. 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:

© Copyright the Linux Foundation 2023. All rights reserved.

```
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
```

8. 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
   ```

9. To correct this deficiency, re-run **aa-genprof** as we did earlier, and in **window2**, ping the **IPv6** loopback and append the additional options

© Copyright the Linux Foundation 2023. All rights reserved.

© Copyright the Linux Foundation 2023. All rights reserved.

Chapter 34

System Rescue

34.1 Rescue Media and Troubleshooting 544
34.2 Using Rescue/Recovery Media 545
34.3 System Rescue and Recovery 546
34.4 Emergency Boot Media 547
34.5 Using Rescue Media 548
34.6 Emergency Mode 550
34.7 Single User Mode 551
34.8 Labs 552

Learning Objectives

By the end of this session, you should be able to:

- Explain what forms system rescue media come in and how they are made available or can be prepared.
- Use rescue media to help repair a system.
- Know how to enter emergency mode and what can be done there.
- Know how to enter single user mode, what can be done there, and how it differs from emergency mode.

34.1 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.

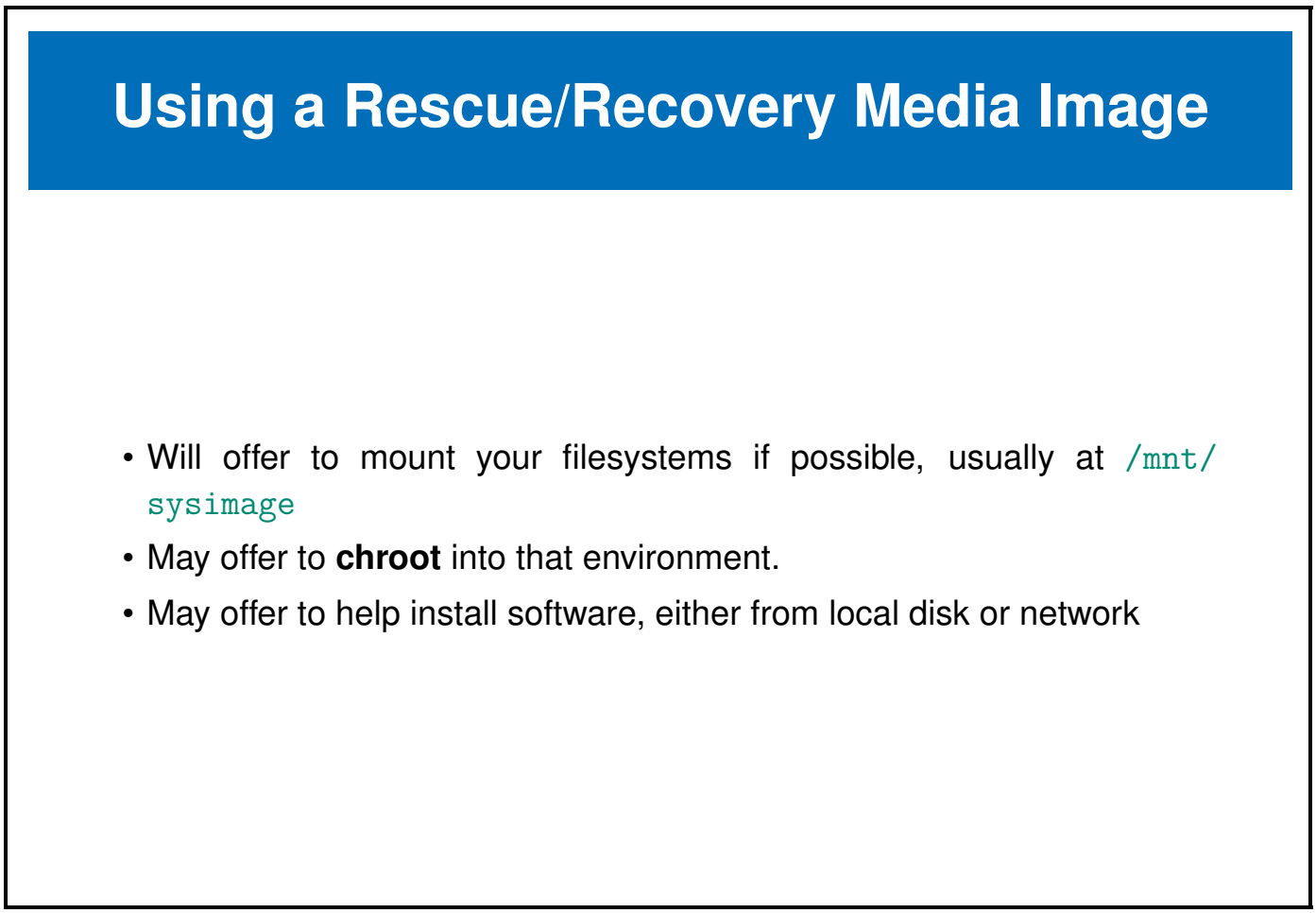
Exact tool choices will vary from one **Linux** distribution to another, but when you boot from an install or live **CD**/**DVD** or **USB** drive you will be able to select an option with a name like `Rescue Installed System`.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

34.2 Using Rescue/Recovery Media



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
```


© Copyright the Linux Foundation 2023. All rights reserved.

34.3 System Rescue and Recovery

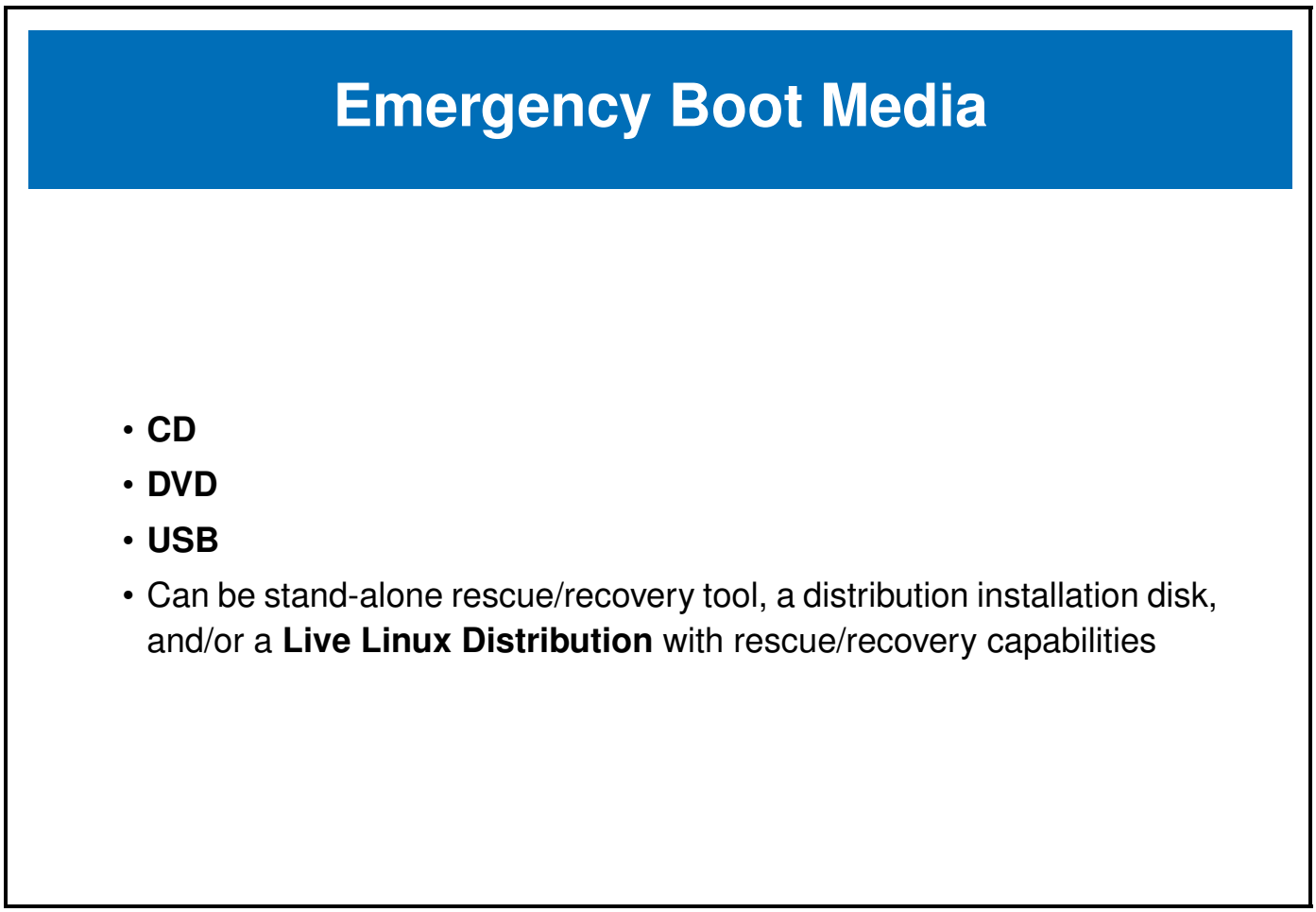
System Rescue and Recovery

- 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.

© Copyright the Linux Foundation 2023. All rights reserved.

34.4 Emergency Boot Media



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.

© Copyright the Linux Foundation 2023. All rights reserved.

34.5 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.

© Copyright the Linux Foundation 2023. All rights reserved.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

34.6 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.

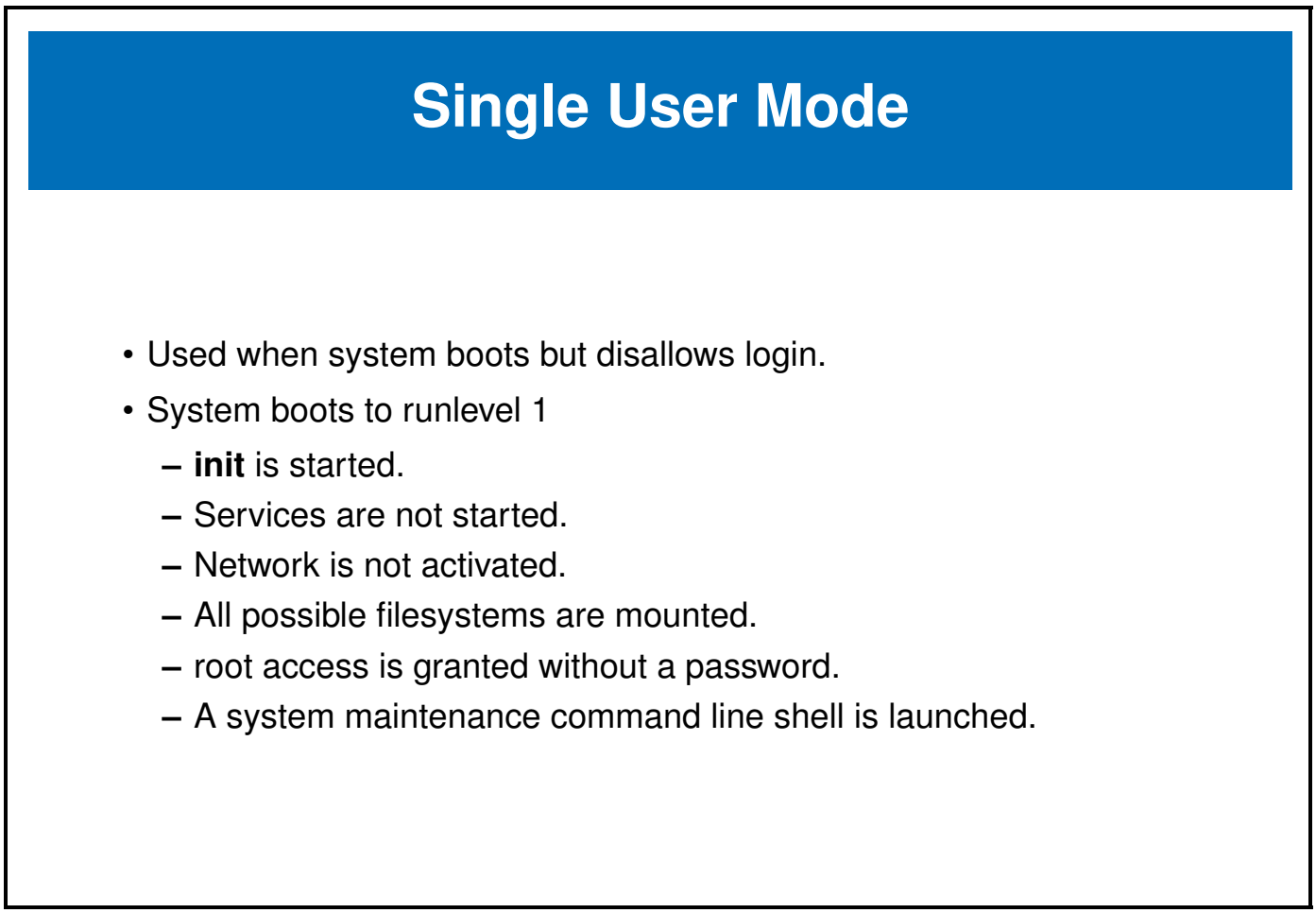
The main advantage of emergency mode over single-user mode (to be described next) is that if **init** is corrupted or not working, you can still mount filesystems to recover data that might be lost during a re-installation.

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.

© Copyright the Linux Foundation 2023. All rights reserved.

34.7 Single User Mode



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`.

© Copyright the Linux Foundation 2023. All rights reserved.

34.8 Labs

Exercise 34.1: 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 34.2: 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 same experience as using rescue media, but it will not always work. For example, if the root filesystem is damaged it will

© Copyright the Linux Foundation 2023. All rights reserved.

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 34.3: 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 34.4: 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
   1+0 records out
   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
   ```

   ```
   -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
   ```

   ```
   1+0 records in
   1+0 records out
   446 bytes (446 B) copied, 0.000124091 s, 3.6 MB/s
   ```


© Copyright the Linux Foundation 2023. All rights reserved.

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 34.5: 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
   ```

   ```
   zsh-5.0.2-7.el7.x86_64
   ```

6. ```
   $ zsh
   ```

   ```
   ....
   [coop@q7]/tmp/LFS201%
   ```


© Copyright the Linux Foundation 2023. All rights reserved.
   ```

Chapter 35

Closing and Evaluation Survey

35.1 Evaluation Survey 555

35.1 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.

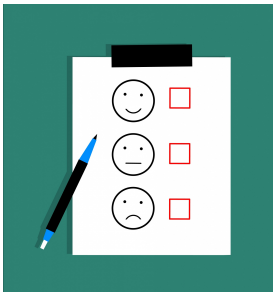


Figure 35.1: **Course Survey**

© Copyright the Linux Foundation 2023. All rights reserved.